Overview | Dataset | Pre-processing | Text analysis

STOCK MARKET

NEWS ANALYSIS

WEB AND SOCIAL MEDIA MINING

Botticelli Tommaso
Cesari Jacopo
Mugenzi Fabrice
Zecchini Giovanni

1 YAHOO FINANCE
Dataset

- Data collection
- Descriptive statistics
- Network Analysis

2 YAHOO FINANCE
Pre-processing

- Text cleaning
- Multi-words detection
- Collocations
- Lemmatization
- Removing stopwords
- Vectorization

3 YAHOO FINANCE
Text analysis

- Explorative data analysis
- Cluster analysis
- Topic Modeling (LSA, NMF, LDA, BTM)
- Topic Coherence Evaluation

4 RETTID - NITTER
Dataset

- Data collection
- Descriptive statistics
- Network Analysis

5 RETTID - NITTER
Pre-processing

- Text cleaning
- Multi-words detection
- Collocations
- Lemmatization
- Removing stopwords
- Vectorization

6 RETTID - NITTER
Text analysis

- Explorative data analysis
- Sentiment analysis

1

YAHOO FINANCE

Dataset

- Data collection
- Descriptive statistics
- Network Analysis

Data collection

The data colletction was conducted on **'Yahoo Finance'** website among different categories :

- stock
- crypto
- private company
- currencies
- treasury bond

Scraping procedure

A nested for loop was implemented to scrape articles from all categories within the period October 1st–20th.

Read more - button

```
read_more_button = WebDriverWait(driver, 3).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, 'button.readmore-button[data-ylk*="readmore"]')))
read_more_button.click()
```

Title

```
title_elem = WebDriverWait(driver, 3).until(
    EC.presence_of_element_located((By.CSS_SELECTOR, 'h1.cover-title')))
```

Article type

```
publishing_div = driver.find_element(By.CSS_SELECTOR, 'div.publishing.yf-m1e6lz')
if 'Yahoo Finance Video' in publishing_div.text:
    is_video = True
    article_data['article_type'] = 'video'
```

Author

```
author_elem = driver.find_element(By.CSS_SELECTOR, 'div.byline-attr-author a.primary-link')
```

Date

```
date_elem = driver.find_element(By.CSS_SELECTOR, 'time.byline-attr-meta-time, time[datetime]')
```

Tickers

```
ticker_elements = driver.find_elements(By.CSS_SELECTOR, 'span.symbol.yf-90gdtp')
```

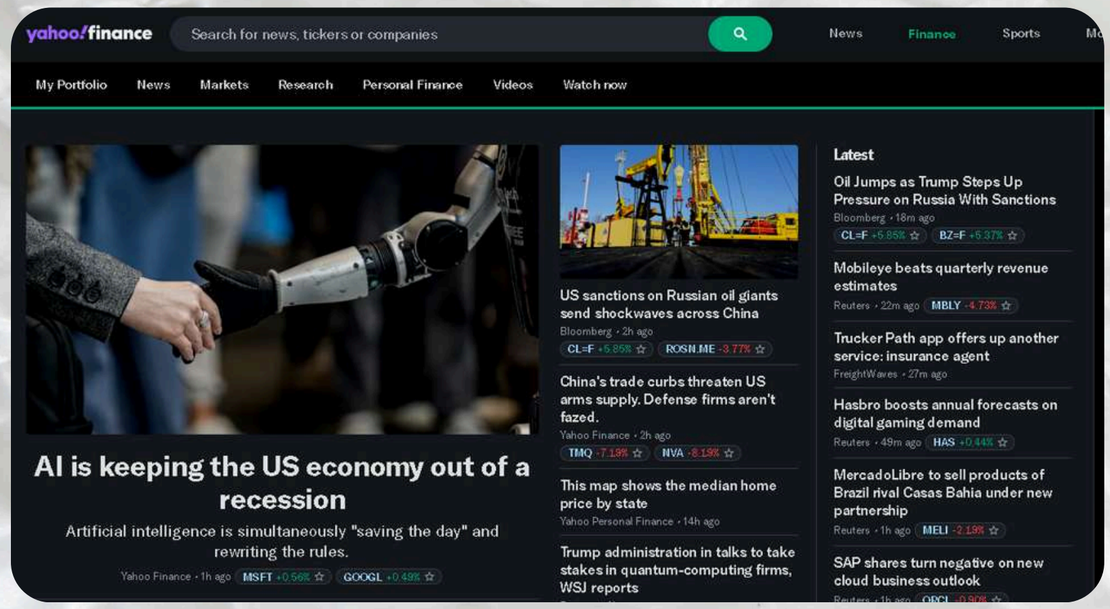
Text

```
paragraphs = driver.find_elements(By.CSS_SELECTOR, 'div.body[data-testid="article-body"] p.yf-1090901')
```

Dataset description

The final dataset consists of **762 articles.**
Total text length: 2'632'297 character
word count: 414'372

- stock - 189
- crypto- 188
- private company - 197
- currencies - 165
- treasury bond - 23

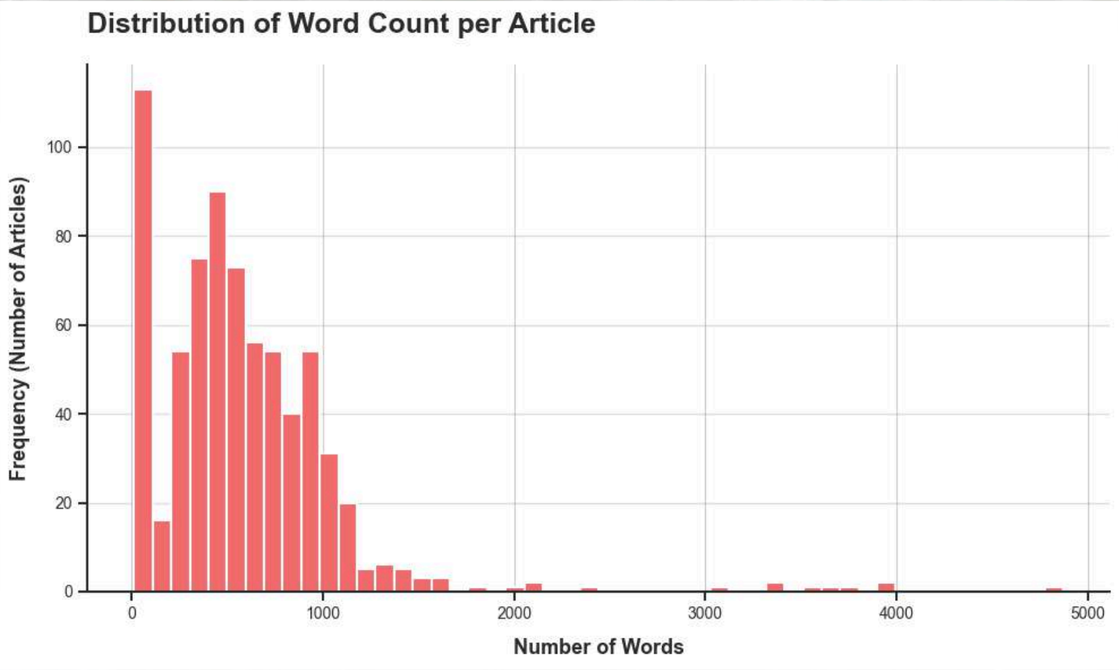


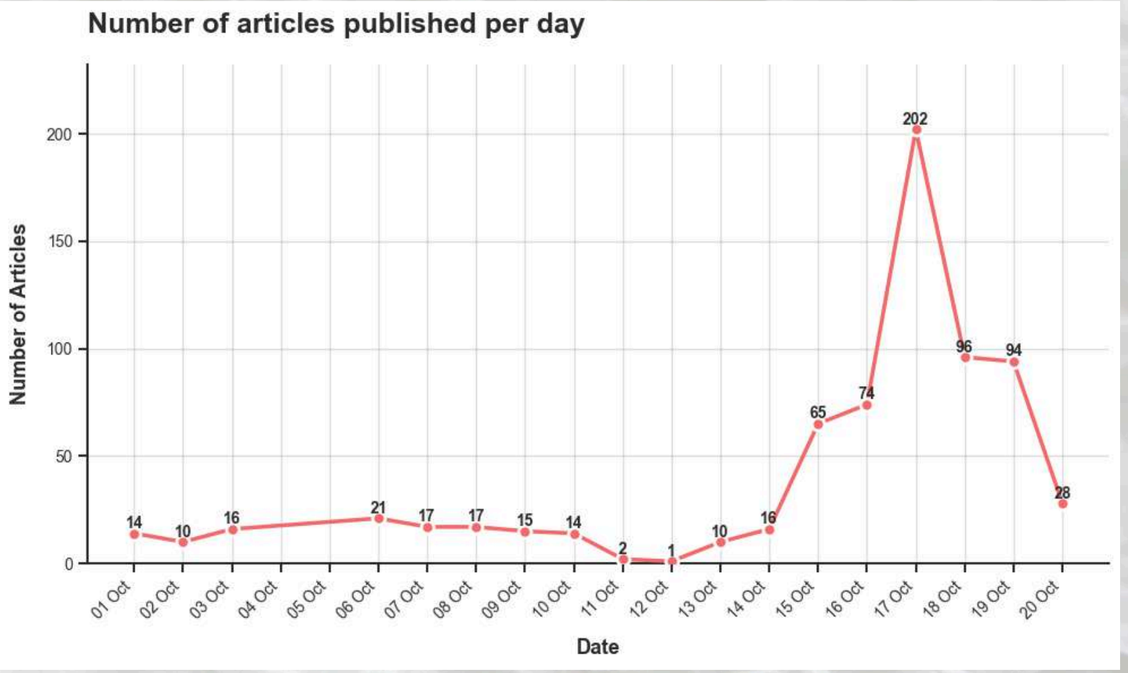
Yahoo Finance dataset	
Variable	**Description**
id	news_id
category	news category (stock, crypto, ...)
title	news title
author	news author
date	news publishing date
parsed_date	datetime format publishing date
text	text of news
url	url of news
tickers	tickers linked to the news

Descriptive statistics

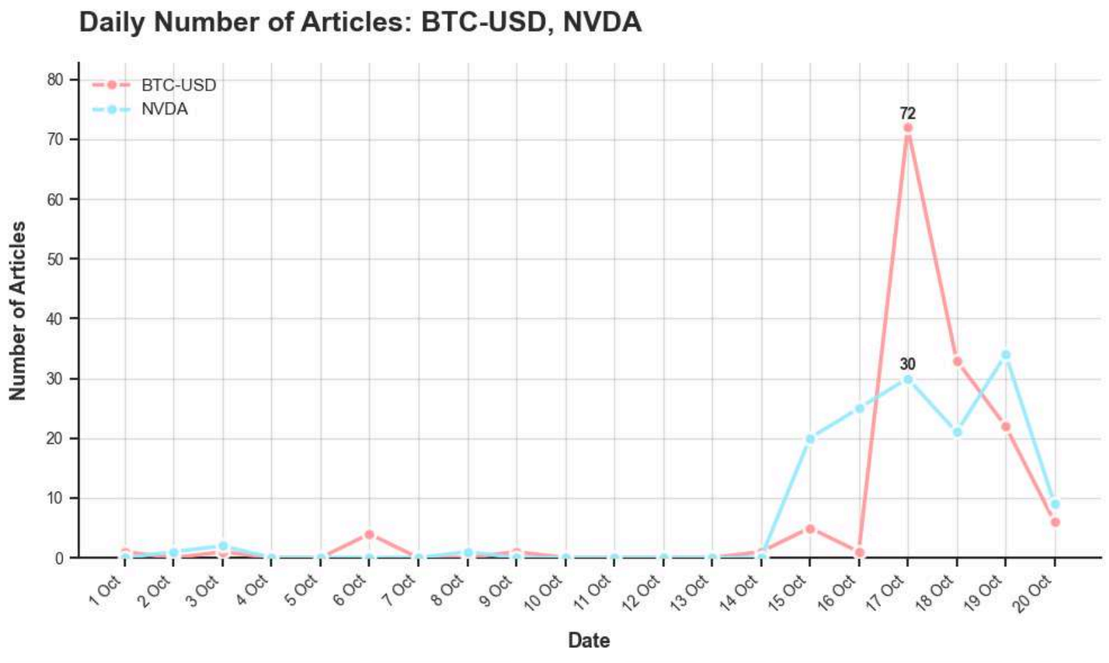
Spike in articles on October 17th, due to the **largest crypto short liquidation in history** and news from Fed Chair Jerome Powell hinting at a **rate cut**.

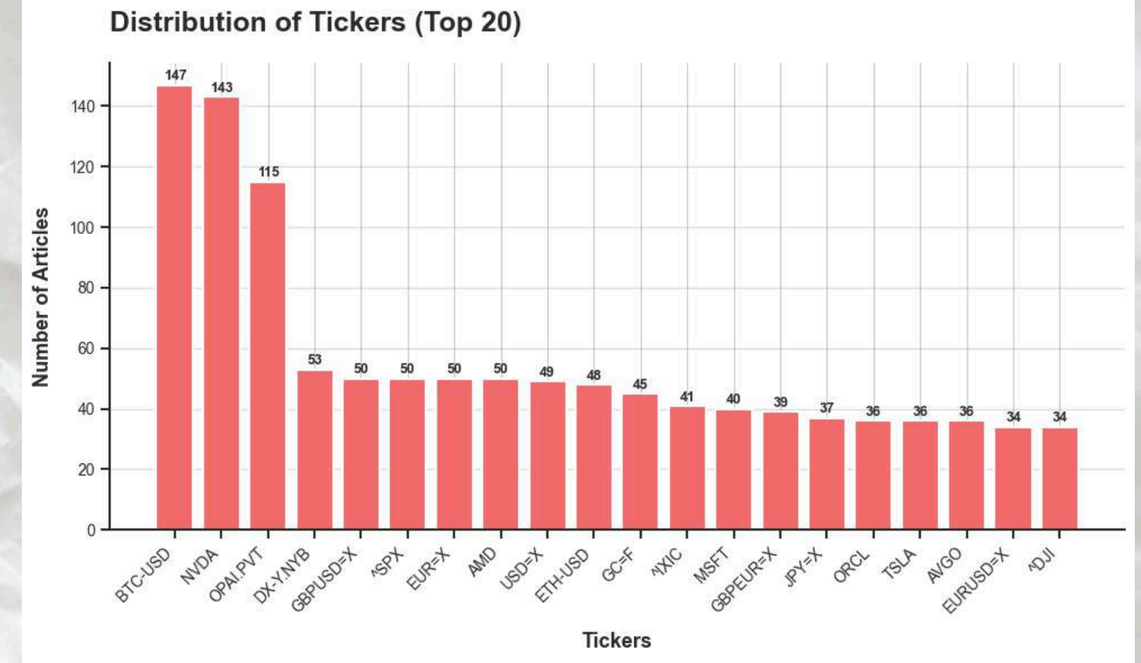
Number of words Statistics					
Min	**1sr Q.**	**Median**	**Mean**	**3rd Q.**	**Max**
12	294	507	582	801	4871





Descriptive statistics - tickers





Network Analysis

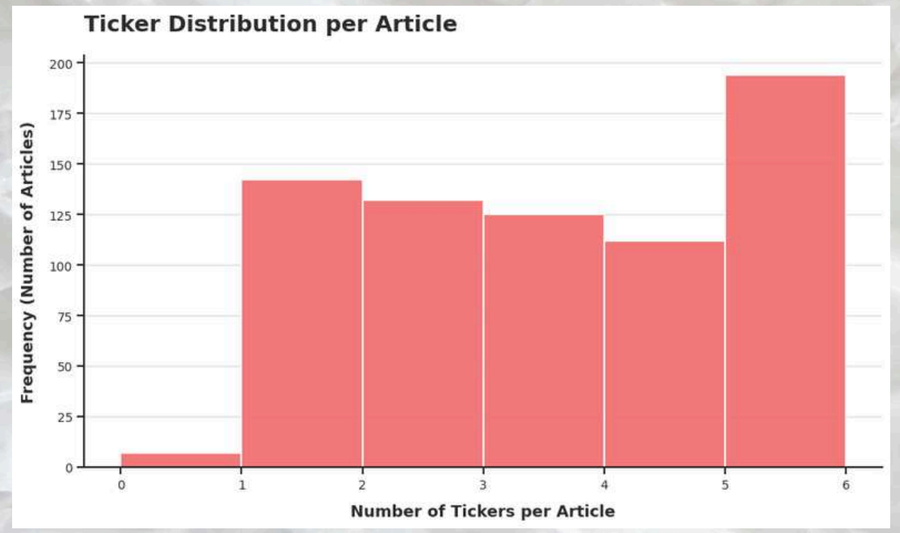
Input:

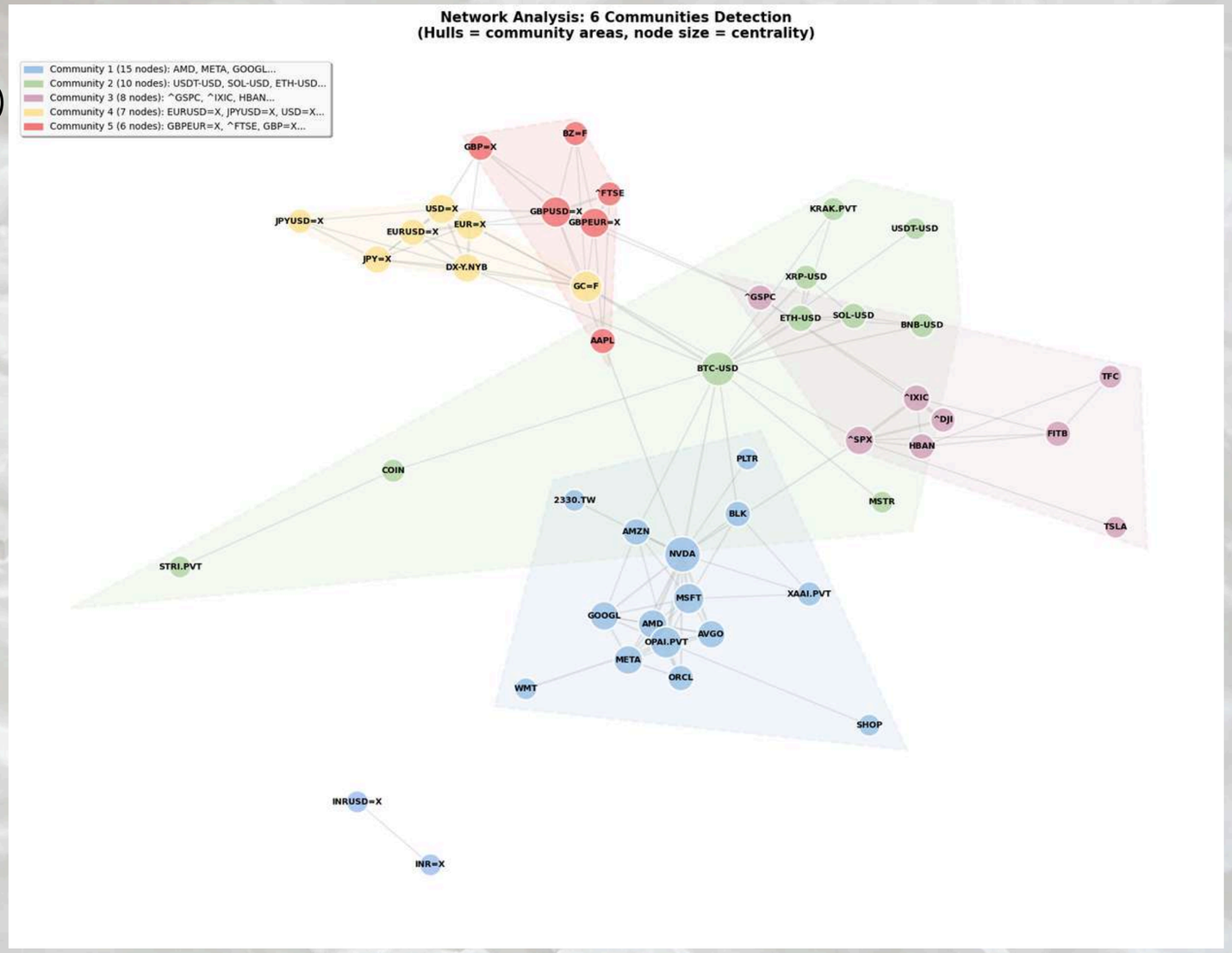
- Ticker × Articles matrix (one-hot encoding)
- **Co-occurrence matrix** (ticker × ticker)

↓

Output:

- Weighted network graph
- **Community detection (Louvain)**
- Metrics: centrality, density, modularity



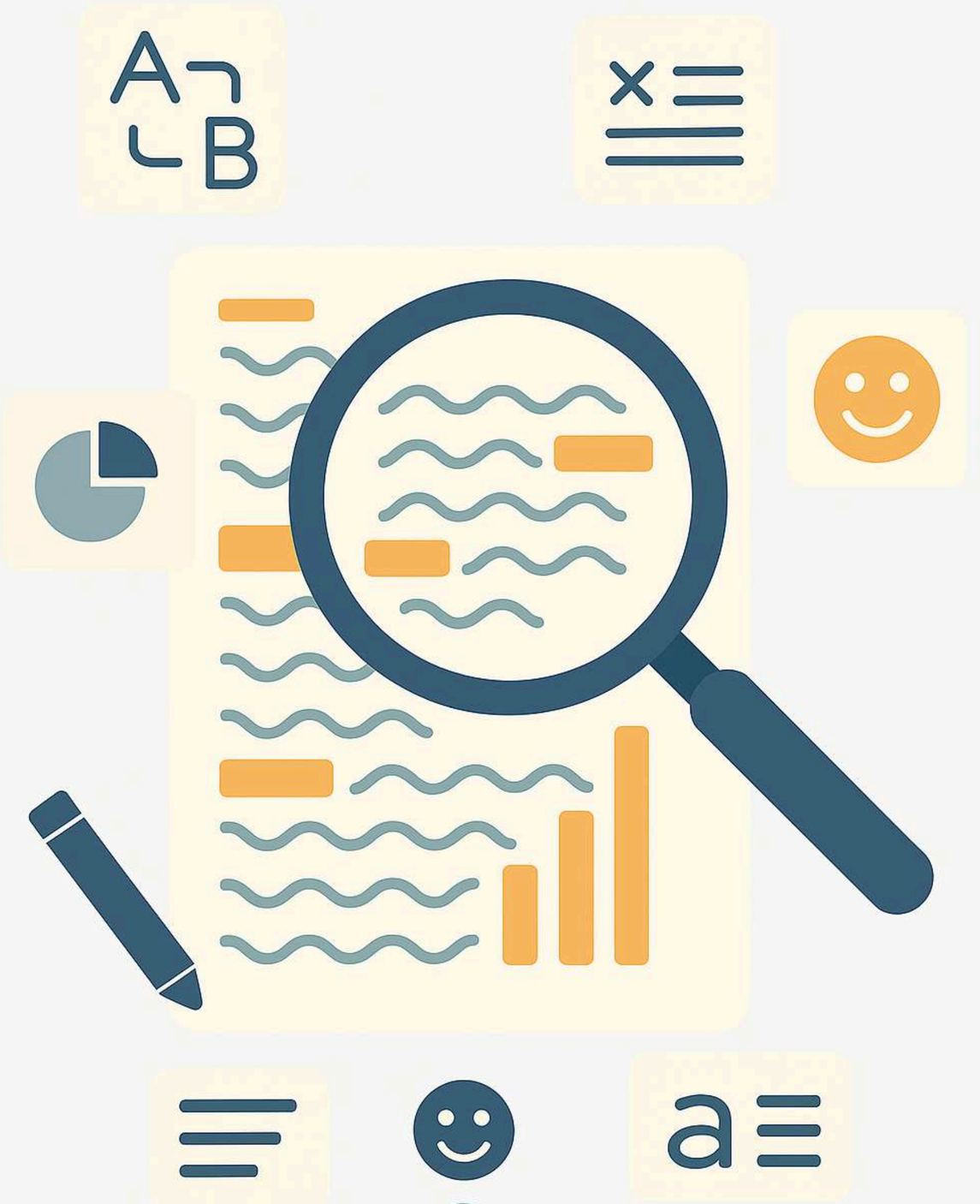


2

YAHOO FINANCE

Pre-processing

- Text cleaning
- Multi-words detection
- Collocations
- Lemmatization
- Removing stopwords
- Vectorization



Text cleaning process:

1. The first cleaning procedure was about, removing eventual record with NAs, or duplicate articles.

2. Secondly we implemented a text cleaning function : **'clean_text'**, to remove : ashgat/mention/punctualization/uppercase

def clean_text(x=None, hashtag=True, mention=True, numbers=True, punctuation=True, lowercase=True):

3. Then we loaded the spaCy model : **'en_core_web_sm'**, which provides the POS tagging and lemmatization required to extract linguistically meaningful bigrams and accurately process text throughout the pipeline.

4. Function: **'my_collocations_POS(texts, nlp, …)'**
The model is used to:

- Annotate each word with its Part of Speech (POS) and lemma.
- Enable the function to filter bigrams that follow specific grammatical patterns (e.g., ADJ–NOUN, NOUN–NOUN, PROPN–PROPN).

5. Function: **'my_collocations(texts, nlp, …)'**

- Annotate the texts to obtain lemmas and POS tags (POS is used only to exclude punctuation, not to filter grammatical patterns).
- Compute unigram and bigram frequencies based on lemmas.
- Calculate the Pointwise Mutual Information (PMI) to identify statistically significant collocations.

6. Function: **'lemmatize_spacy_en()'**

- Tokenizes the text into individual words.
- Lemmatizes each word, reducing it to its base form.
- Identifies and removes stopwords, such as "the", "is", and "and".
- This process ensures that the text is standardized and ready for consistent linguistic and statistical analysis.

7. Function: **'create_term_document_matrix()'**

- Rows represent unique terms in the vocabulary.
- Columns represent individual documents.
- Values indicate how many times each term appears in each document.
- This matrix forms the foundation for subsequent analytical steps, such as topic modeling, clustering, or sentiment analysis.

3

YAHOO FINANCE

Text Analysis

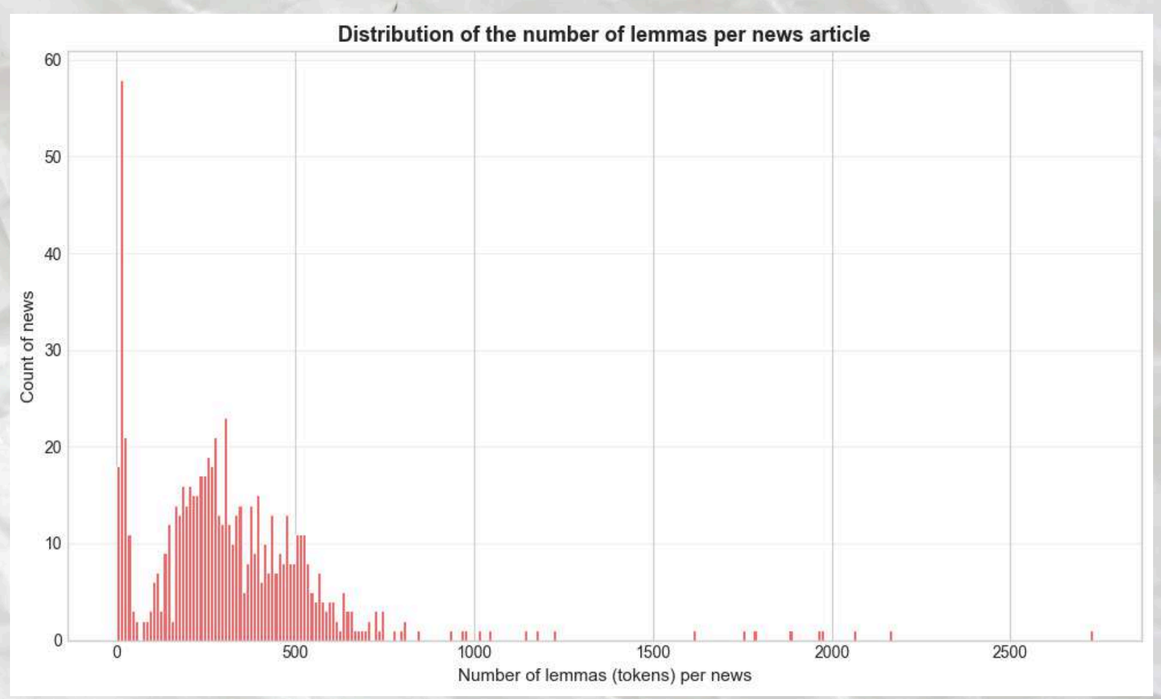
- Explorative data analysis
- Cluster Analysis
- Topic Modeling (LSA, NMF, LDA, BTM)
- Topic Coherence Evaluation

1/4 EDA - Lexicometric measures

Lexicometric measures					
N	**V**	**H**	**TTR**	**HTR**	**TMF**
228795	26349	14876	0.115	0.564	8.683

- **N**: corpus size, given by the total number of occurrences
- **V**: vocabulary size, number of unique terms
- **H**: hapax, terms occurring only once
- **TTR** = V/N Type-Token-Ratio
- **HTR** = H/V Hapax-Type-Ratio
- **TMF** = N/V Type mean frequency (measure of redundancy)

Summary of number of lemmas					
Min	**1sr Q.**	**Median**	**Mean**	**3rd Q.**	**Max**
7	172	280	321.34	438	2722

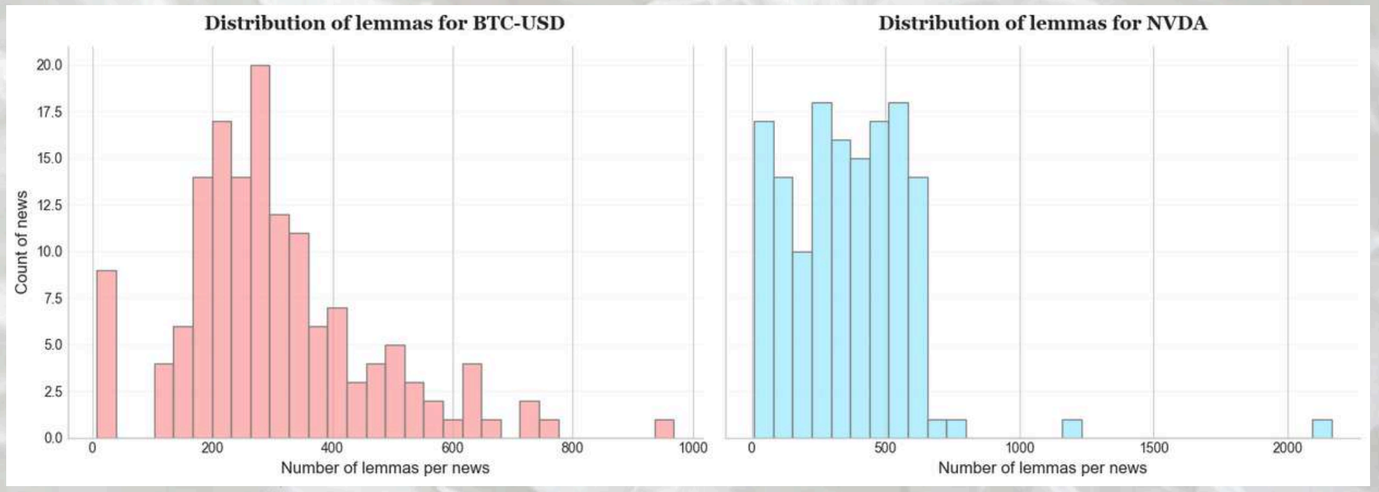


2/4 EDA - Lexicometric measures for subcorpora

The sparsity is quite high at 56%, meaning many words are specific to a single ticker and not shared across documents

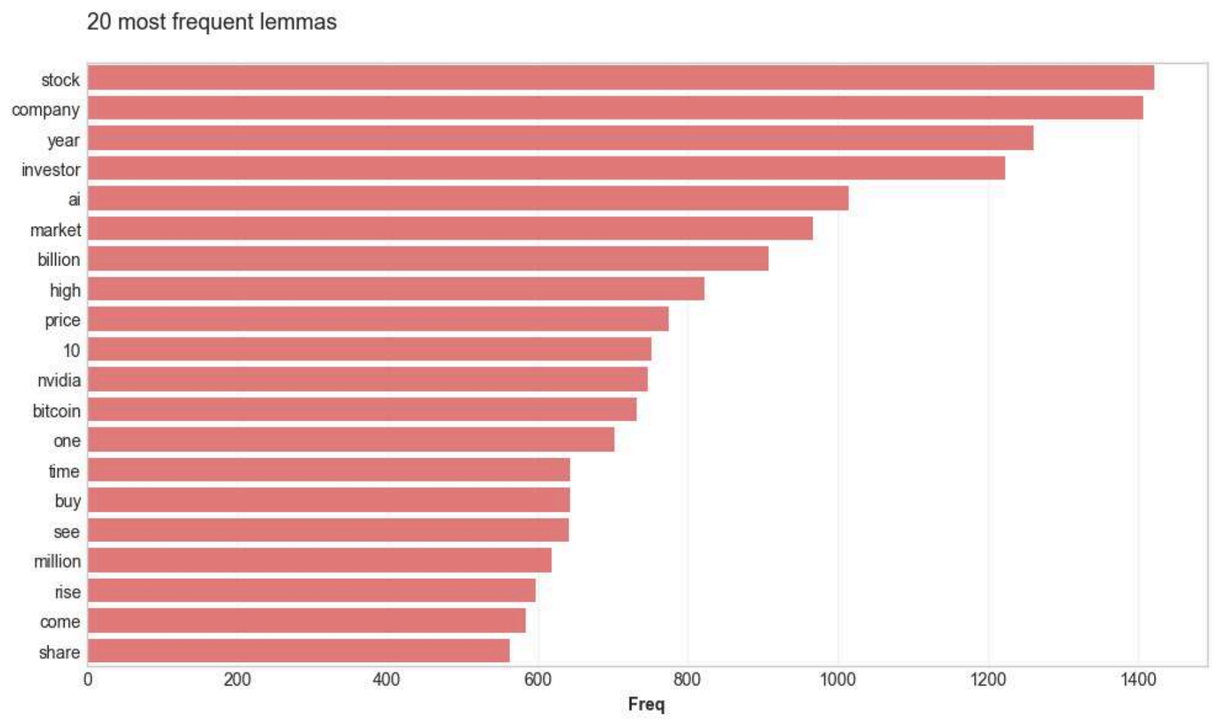
```
Non-/sparse entries: 18195/23100
Sparsity           : 56%
Maximal term length: 33
Weighting          : term frequency (tf)

Sample:
ticker     BOTH  BTC-USD  NVDA
investor     74      235   393
bitcoin      37      653     5
nvidia       16       17   643
stock        19      141   492
company      21      154   464
ai           11       53   445
billion      22      182   302
year         10      161   333
10           17      138   261
market       12      228   160
```

```
$BTC-USD
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8.00   215.50   275.00   308.79   381.50   996.00

$NVDA
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9.00   175.00   374.00   368.56   526.50   2208.00
```



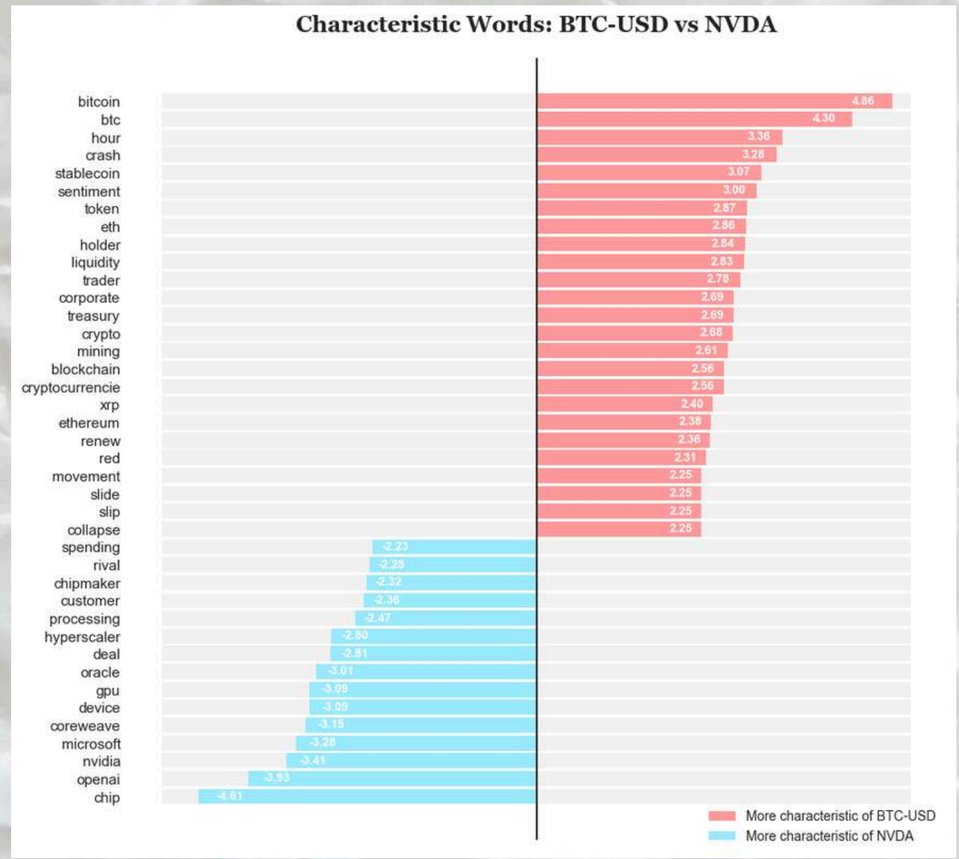
3/4 EDA - Lexicon analysis

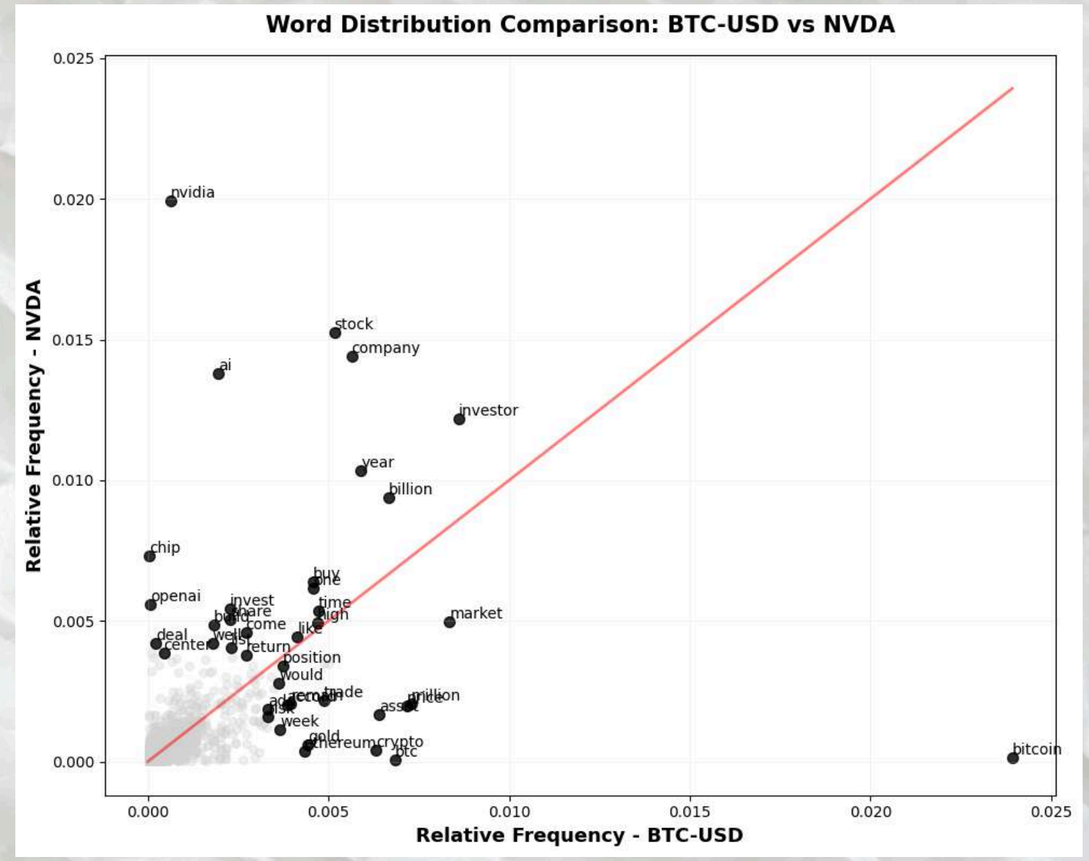
In this part the aim is to understand what the corpus is about, basically which terms (lemmas) are most present.



4/4 EDA - BTC vs NVDA

The graph shows which words are more or less likely to come from one of the two tickers.
The first one uses the **log odds ratio**, while the second one uses **frequency**



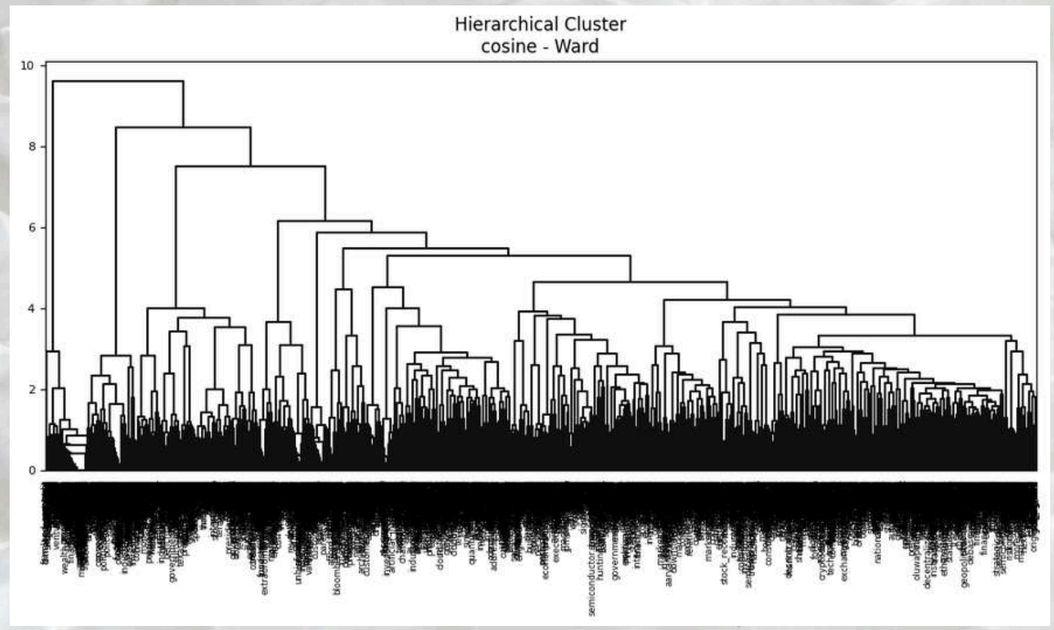


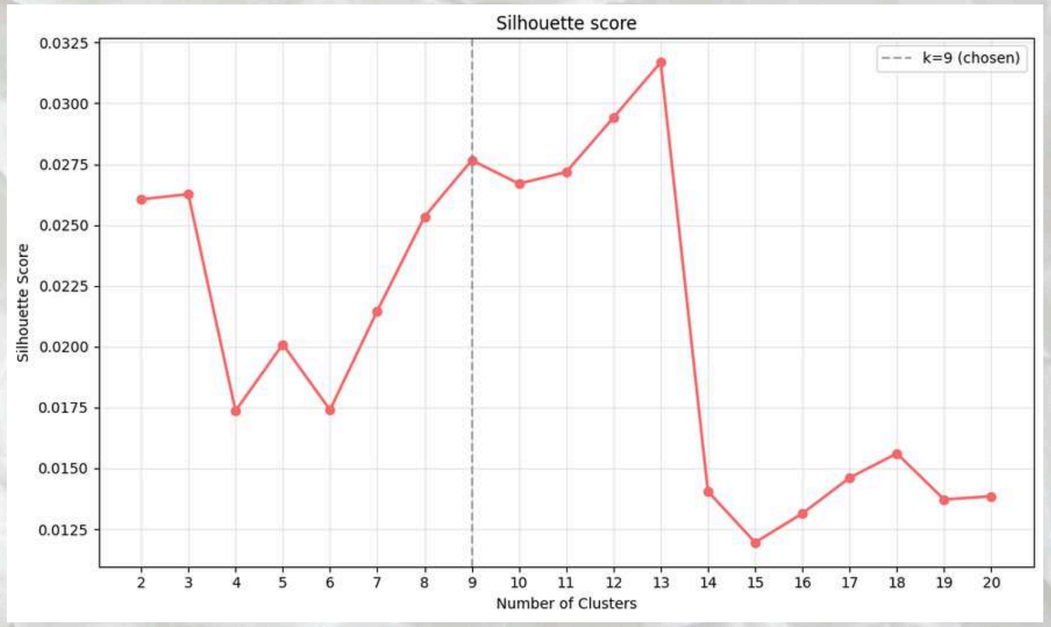
1/3 Cluster analysis - terms

- **Goal**: Identify groups (clusters) of words that are closely associated with each other

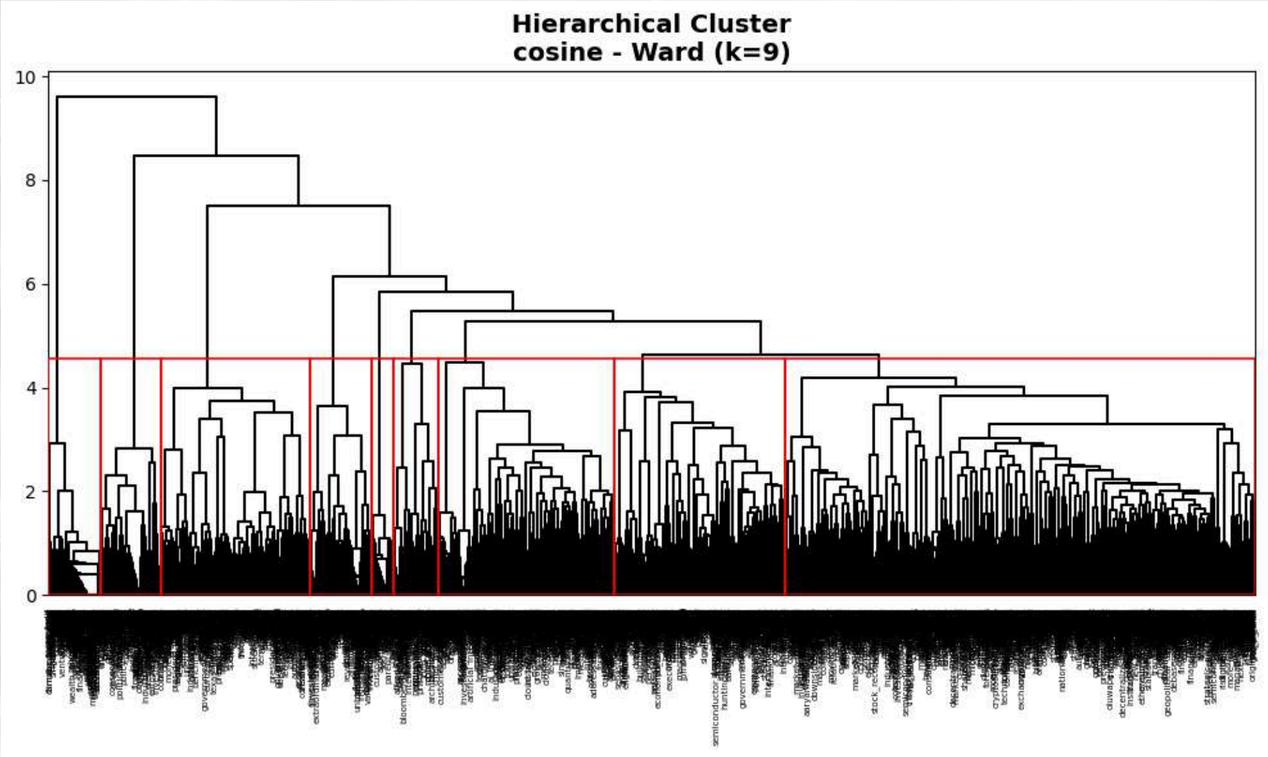
Step :

1. **Create Term-Document Matrix (TDM):**
   - Reduced by keeping only terms that appear in at least 1% of documents
2. **Similarity Measure: Cosine Distance**
   - more suitable for text data with varying document lengths
3. **Clustering Method: Ward's hierarchical clustering**
   - Minimizes within-cluster variance





2/3 Cluster analysis - terms

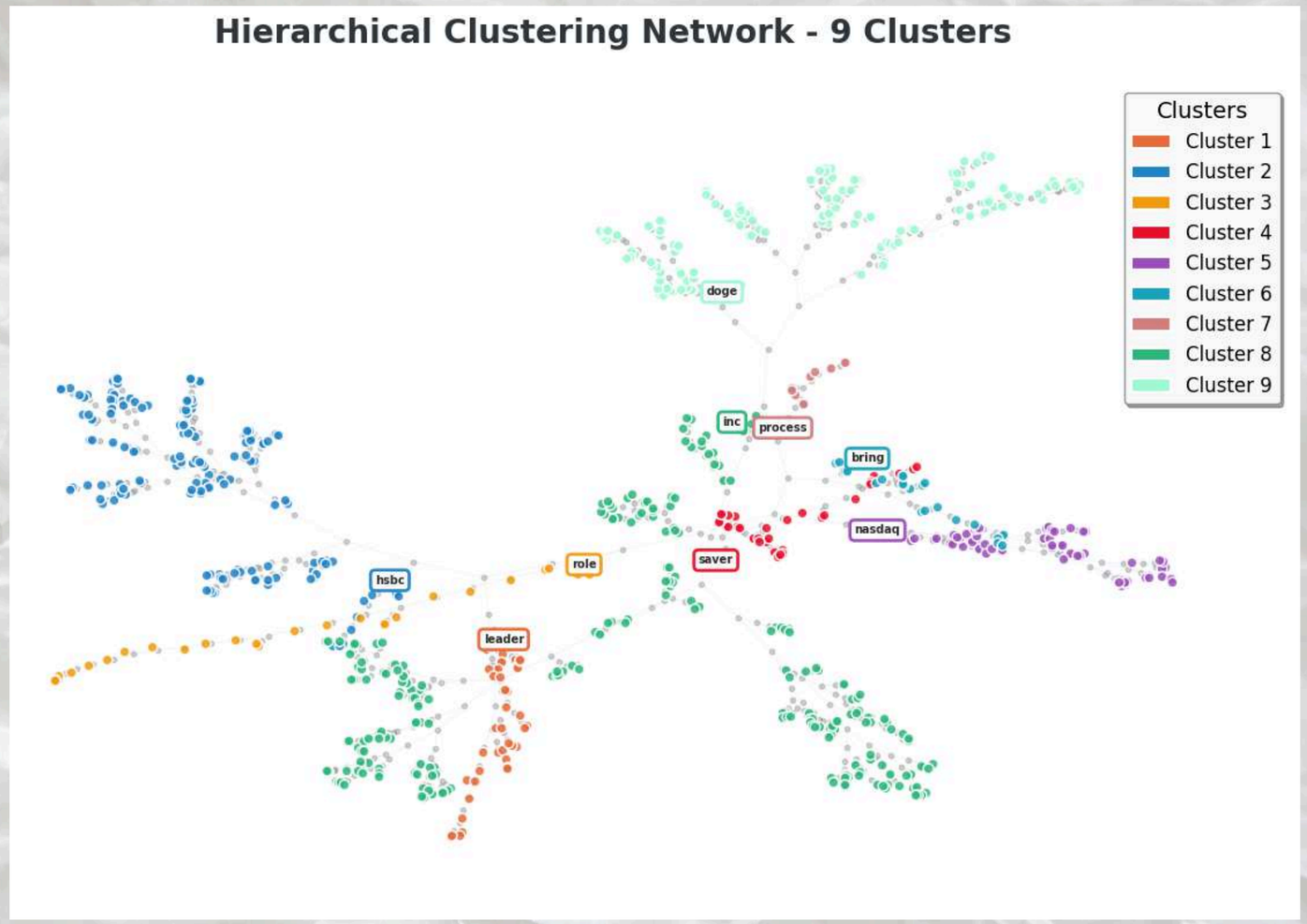


clust	terms
1	hold (199); asset (189); fund (183); value (168); start (149)
2	expect (206); become (180); keep (161); dollar (157); likely (150)
3	market (375); year (356); high (331); company (323); see (296)
4	long (275); may (267); move (260); note (227); risk (203)
5	cut (203); announce (163); plan (149); part (126); let (90)
6	china (112); behind (91); currency (81); bloomberg (61); equity (61)
7	investor (337); one (306); stock (279); buy (277); time (274)
8	read (258); take (239); use (236); even (207); however (199)
9	million (217); report (217); base (204); suggest (164); point (148)

3/3 Cluster analysis - terms

Filters to improve Visualization :

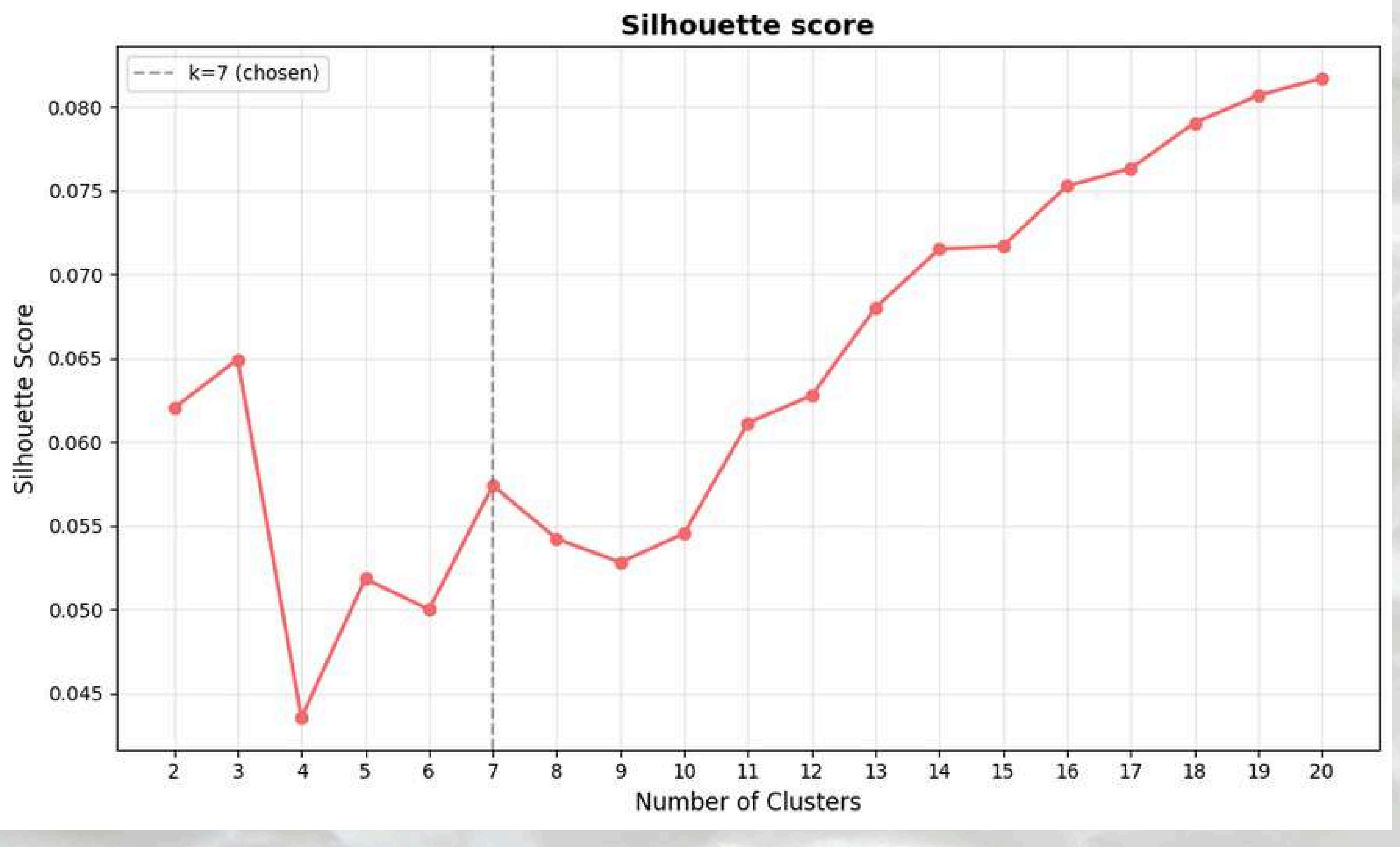
1. **Frequency filter**: Dimensionality reduction of the term-document matrix
Removed terms that are too rare (< 25th percentile) or too common (> 95th percentile)
Objective: retain only informative terms for cluster discrimination

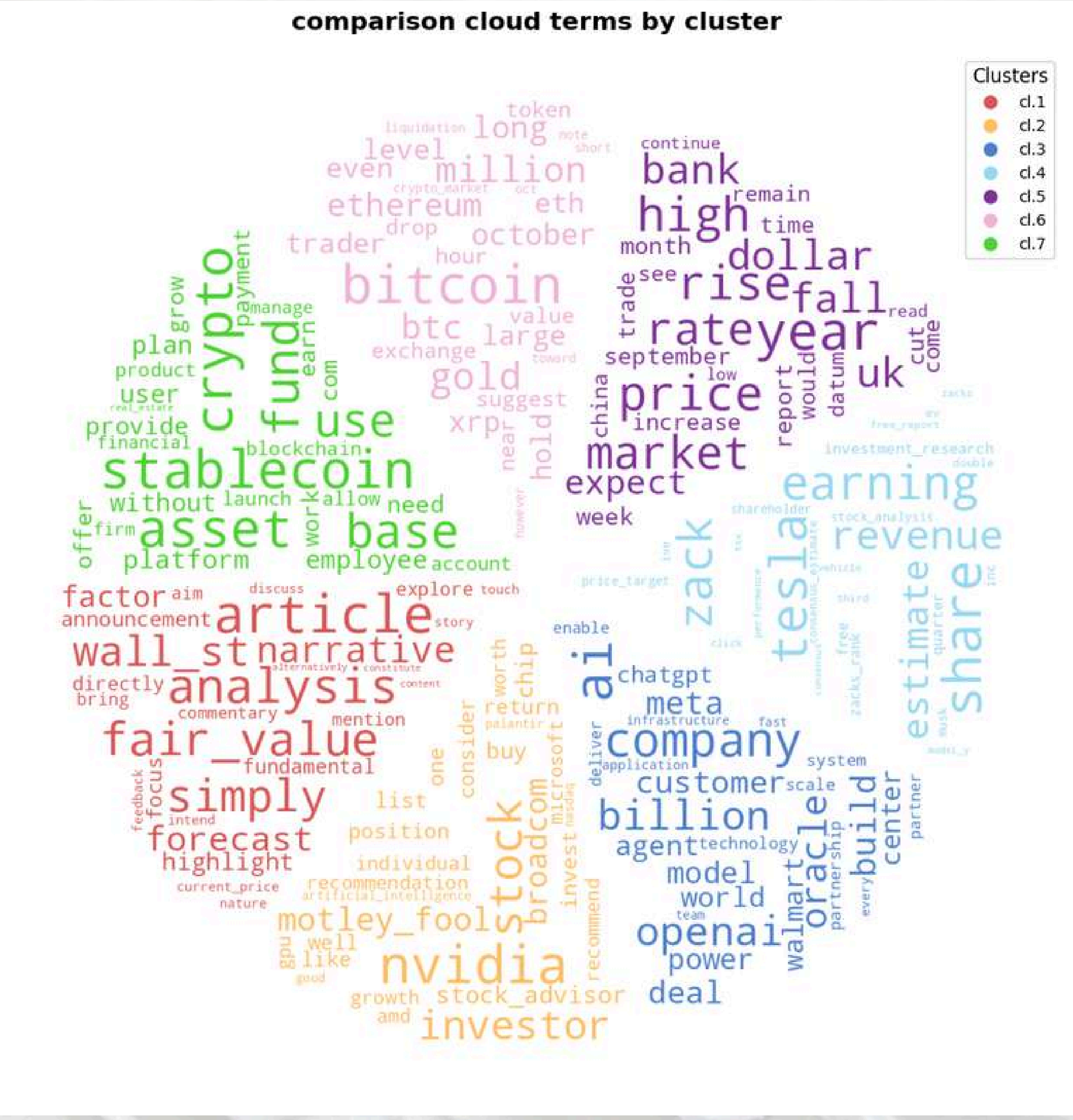


Cluster analysis - documents

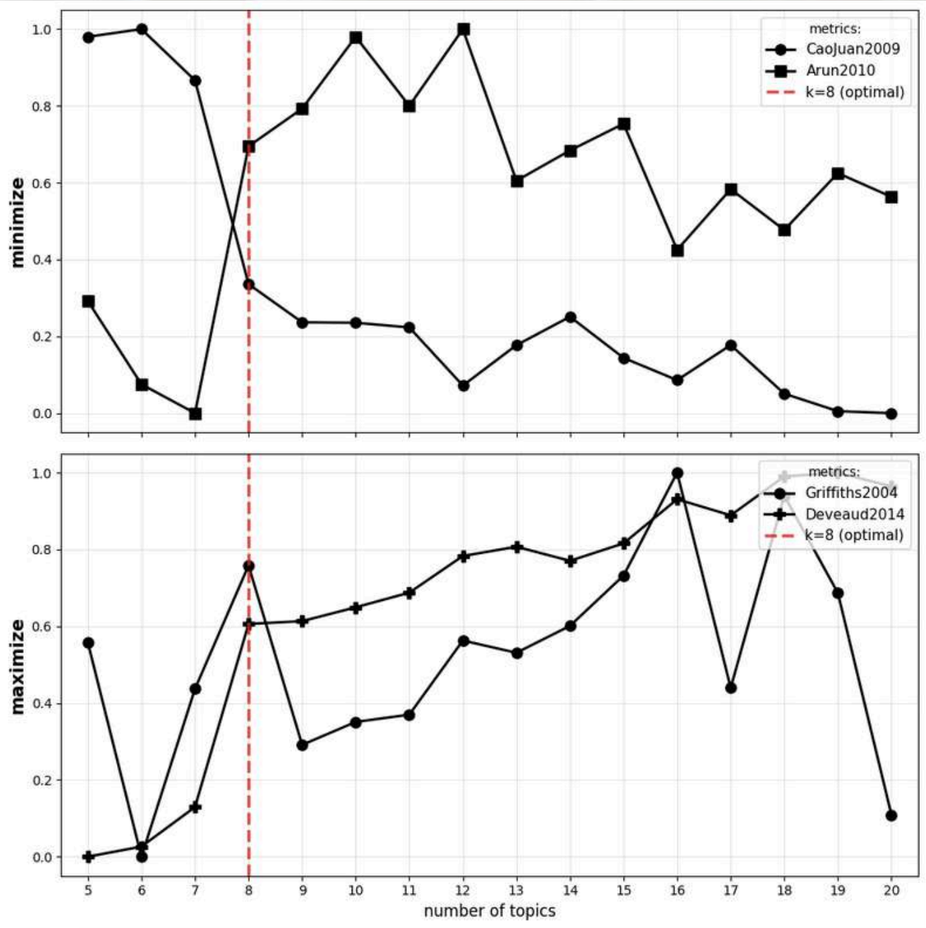
Another way to approach the problem is to **group documents** according to the terms they contain and then identify which terms characterise the document clusters

Optimal Number of Cluster Selection
To achieve the best trade-off between topic coherence and interpretability, we selected k = 7 as the optimal number of topics.





1/4 Topic Modeling - optimal n. of topic



1. Evaluation of the Number of Topics (k)
The optimal number of topics (k) was determined by systematically **evaluating LDA** models with k ranging from **5 to 20.**
We applied four complementary metrics to assess model quality:

- **Griffiths2004** (Griffiths & Steyvers, 2004) – maximizes log-likelihood
- **CaoJuan2009** (Cao et al., 2009) – minimizes cosine similarity between topics
- **Arun2010** (Arun et al., 2010) – minimizes symmetric KL divergence
- **Deveaud2014** (Deveaud, San Juan & Bellot, 2014) – maximizes Jensen-Shannon divergence

◦ Based on the four indices, we selected **8 topics** as the optimal number for the subsequent analysis

2/4 Topic Modeling - coherence evaluation

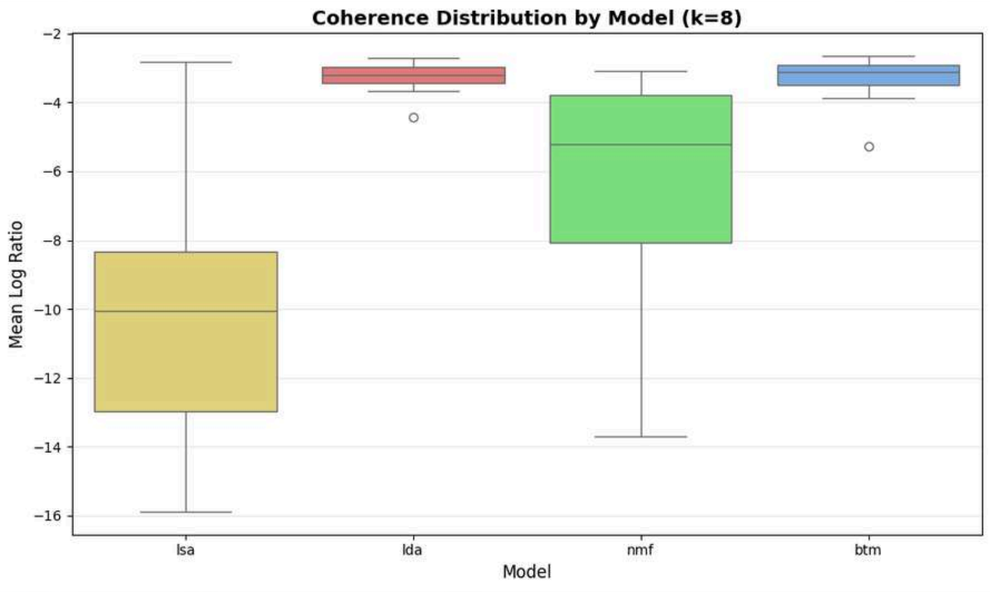
2. Model Fitting

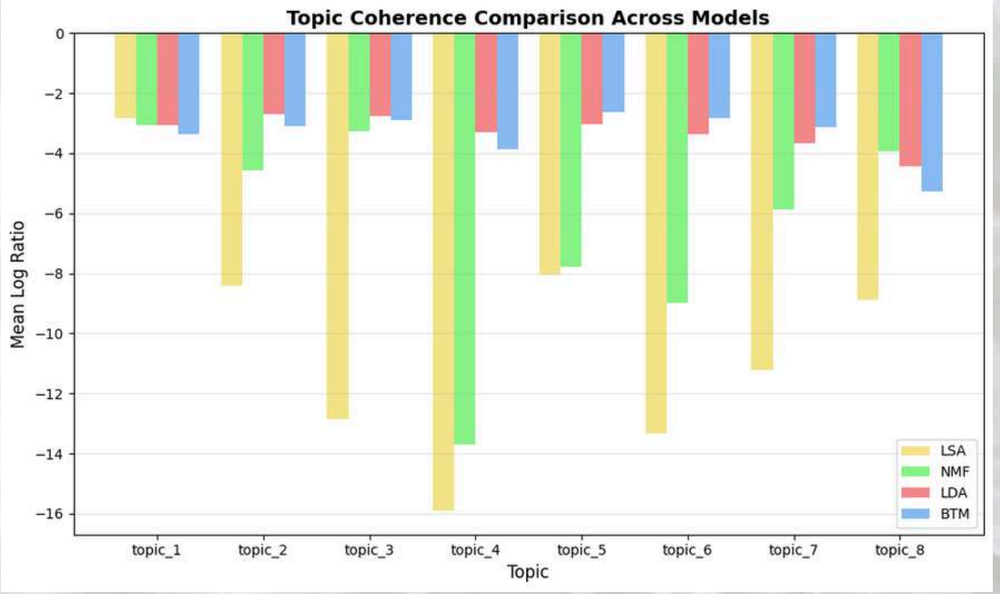
We evaluated 4 topic modeling algorithms : LSA, NMF, LDA, and BTM, each with k = 8 topics.

Objective: Assess the semantic interpretability of topics beyond mathematical optimality.

3. Topic Coherence Evaluation

- Coherence measured using the ***mean_logratio metric*** → evaluates term co-occurrence probability within each topic.
- Topic coherence computed for each model
- **Box plots** generated to summarize and compare the models' statistical performance

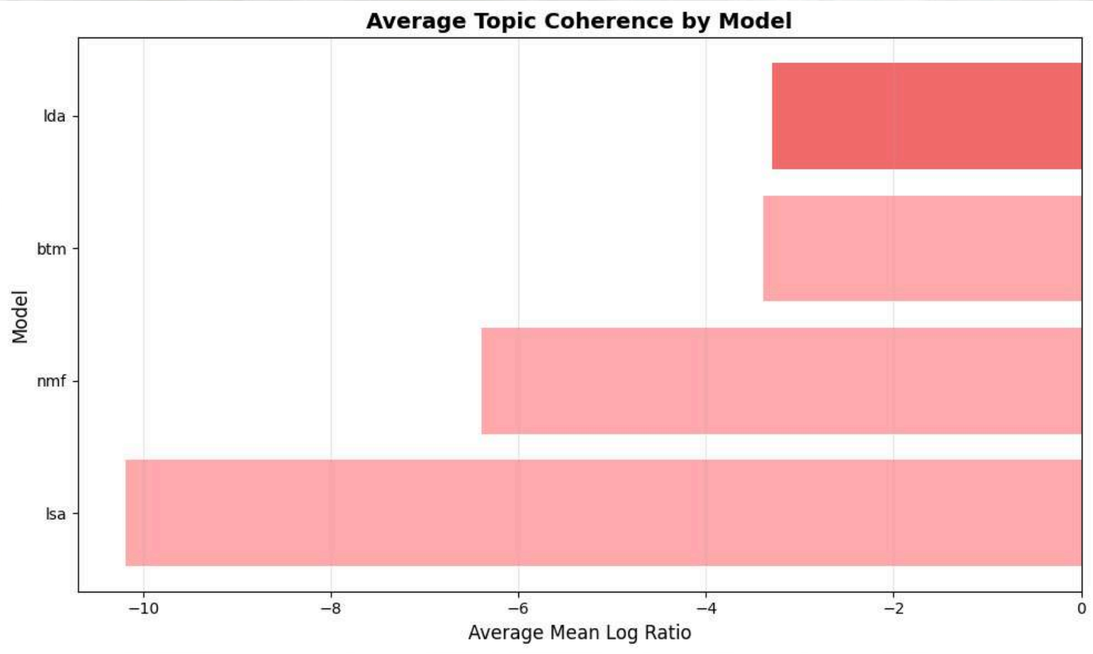


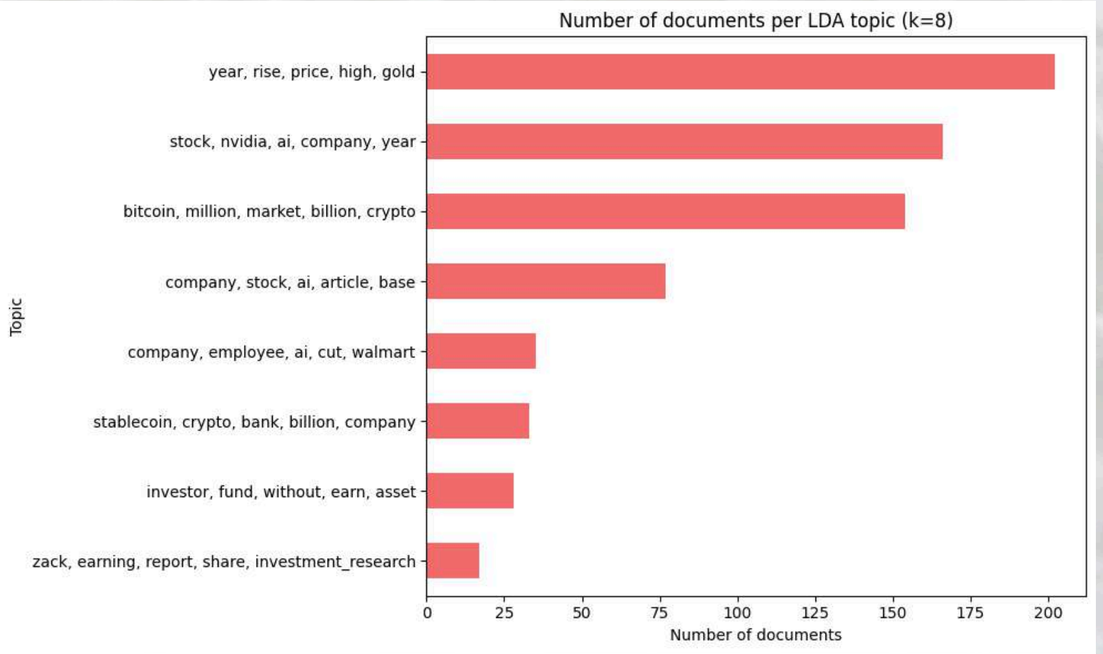


3/4 Topic Modeling - LDA

4. Conclusion - Model choice

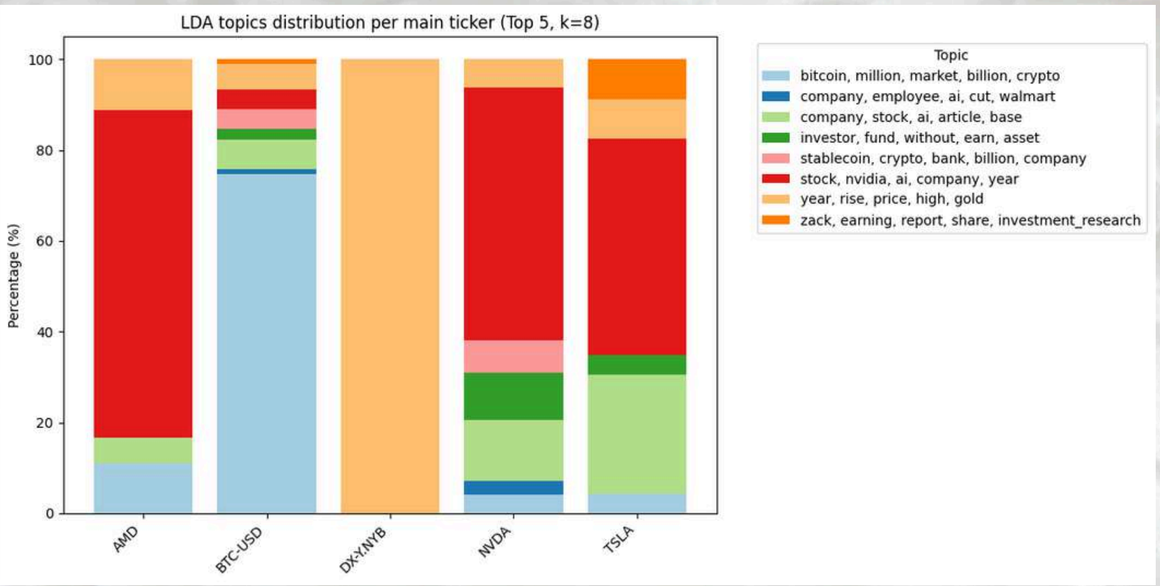
- Both **BTM and LDA** models **show similar behavior** in terms of topic coherence across the eight topics.
- The **LDA model was retained**, as it slightly outperformed BTM.
- According to theory, LDA is also more appropriate for medium-length documents, such as those in our dataset.

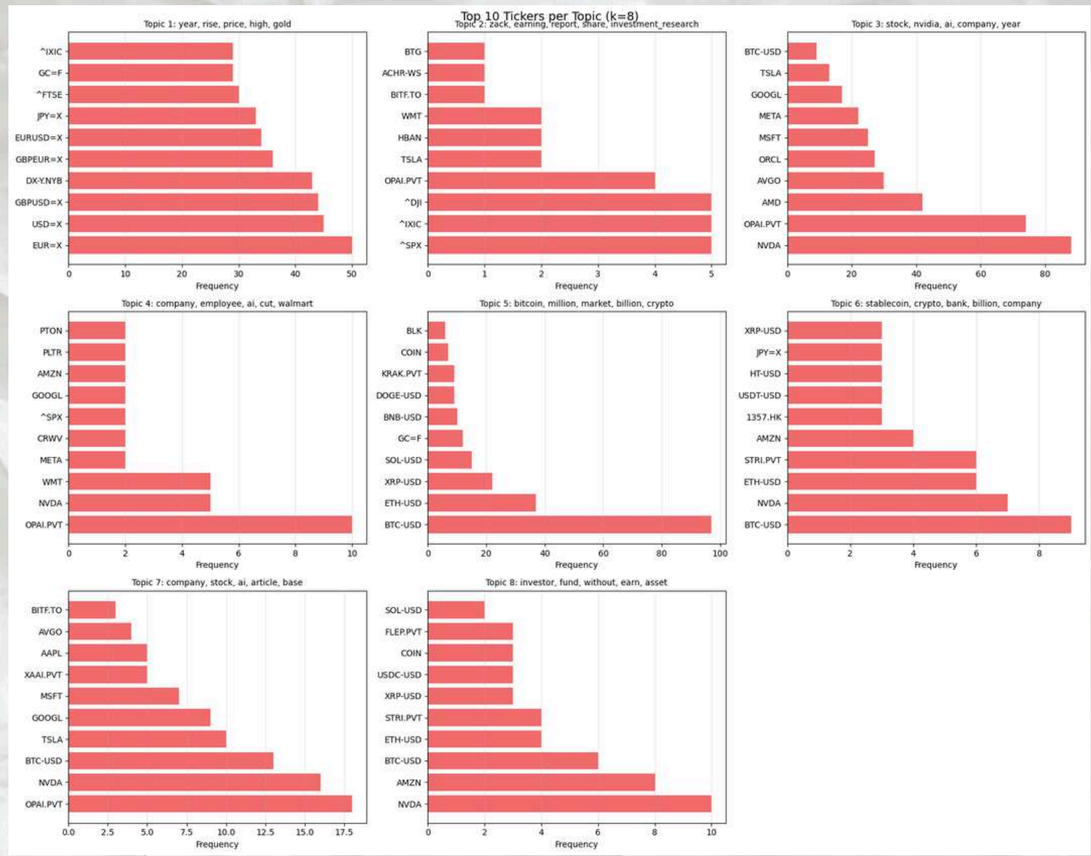




4/4 Topic Modeling - plots

5. **Plots** :
- Topic distribution per tickers
- Top 10 tickers per topic

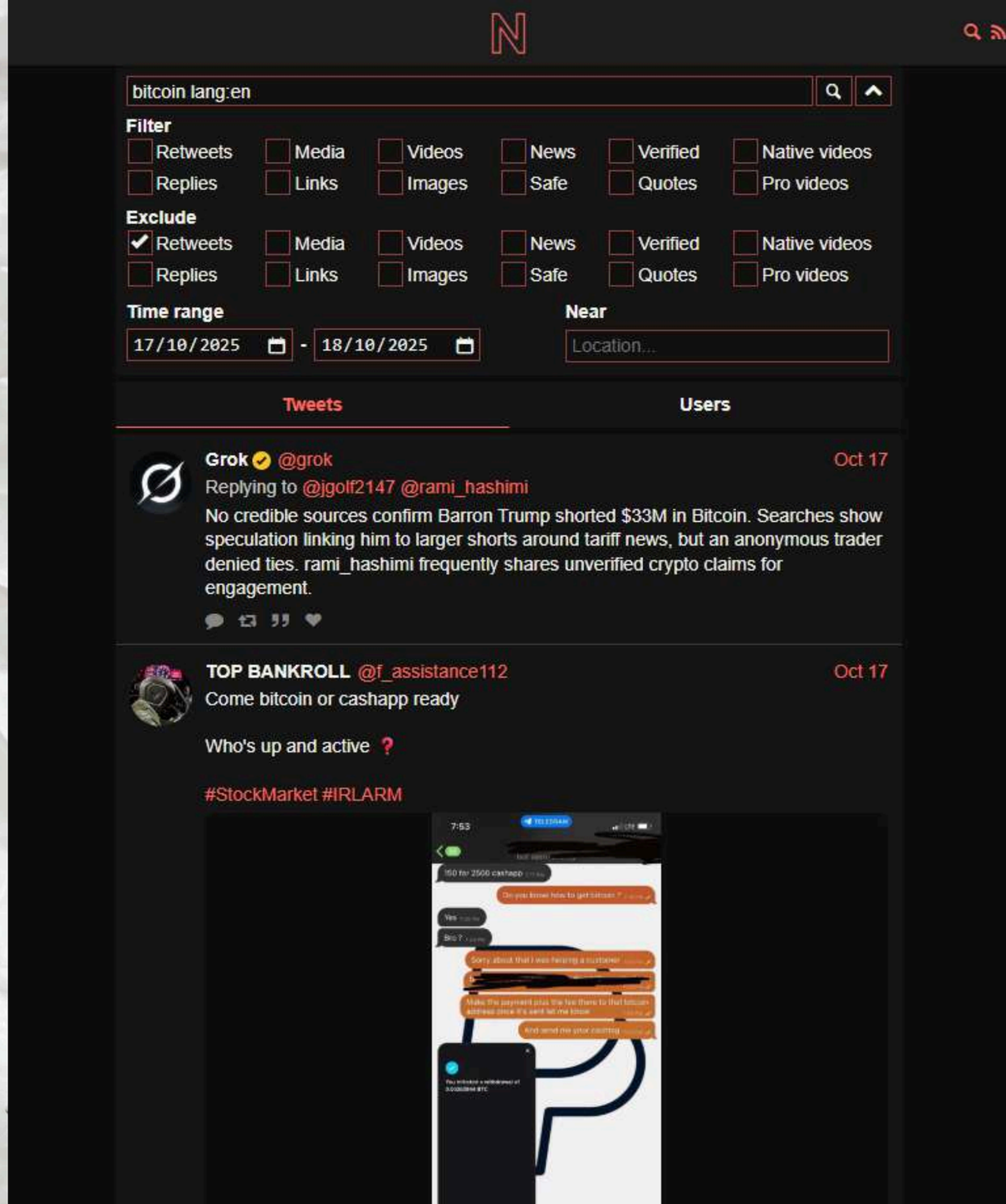




4

NITTER
Dataset

- Data collection
- Descriptive statistics
- Network Analysis



Data collection

The data colletction was conducted on **‘Nitter’** website among two different keywords: **Bitcoin**, **Nvidia**

Scraping procedure

A nested for loop was implemented to scrape 2500 tweets from each keyword for both 17th and 18th of October .

Selenium

```
# Find the div.show-more
load_more_div = driver.find_element(By.CSS_SELECTOR, "div.show-more:not(.timeline-item)")
load_more_link = load_more_div.find_element(By.TAG_NAME, "a")

link_text = load_more_link.text.strip()
if "Load more" in link_text:
    driver.execute_script("arguments[0].scrollIntoView();", load_more_link)
    time.sleep(1)
    load_more_link.click()
```

URL

```
# URL della pagina con le date parametrizzate
url = f"https://nitter.net/search?f=tweets&q=bitcoin%20lang%3Aen&e-nativeretweets=on&since={since_date}&until={until_date}&near="
```

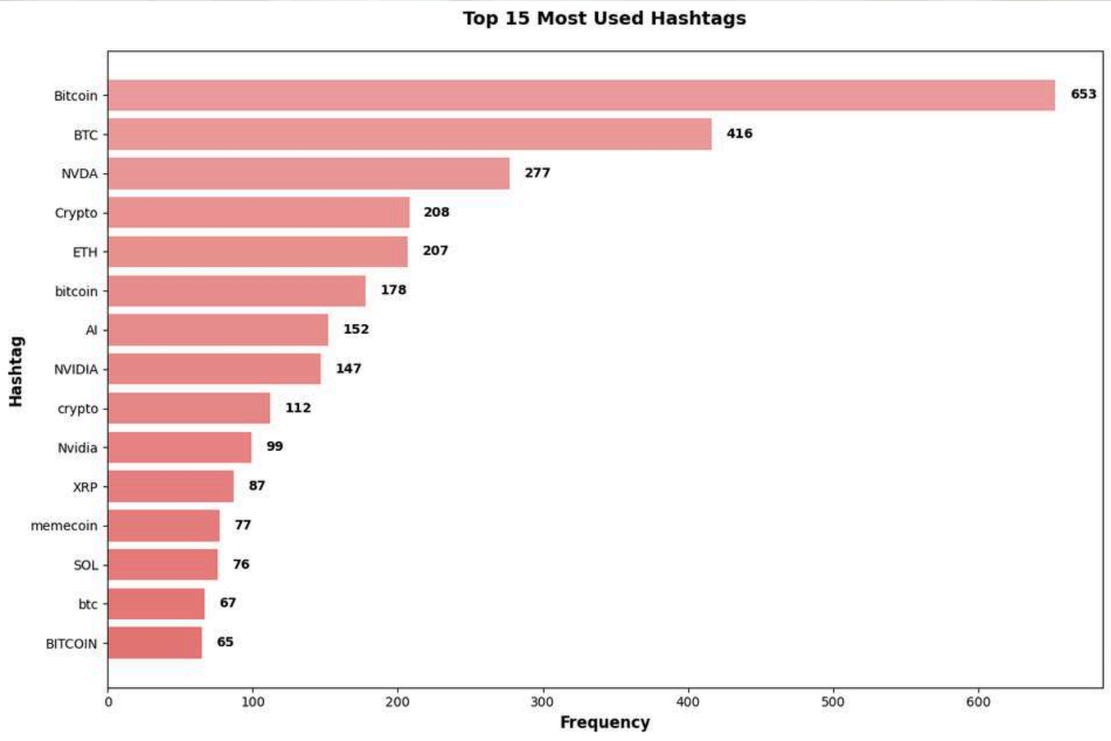
The dataset

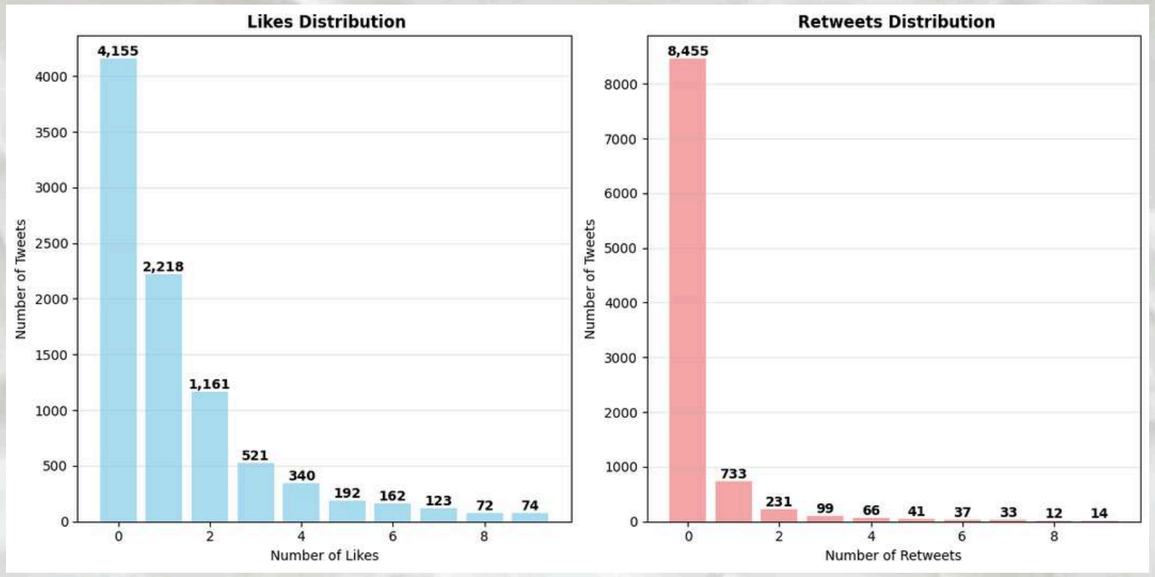

```python
tweets_data.append({
    'thread_title': None,
    'thread_author': None,
    'thread_score': None,
    'thread_num_comments': None,
    'text_id': tweet_id,
    'comment_parent_id': None,
    'text_author': username,
    'text': tweet_text,
    'likes': likes,
    'text_date': tweet_date,
    'text_num_replies': comments,
    'retweets': retweets,
    'comment_parent_author': None,
    'text_mentions': ' '.join(mentions) if mentions else '',
    'text_hashtags': ' '.join(hashtags) if hashtags else '',
    'argument': 'Bitcoin',
    'site':'Nitter'
})
```

```python
# Extract mentions
mentions = re.findall(r'@(\w+)', tweet_text)
mentions = [m for m in mentions if m != username]

# Extract tags
hashtags = re.findall(r'#(\w+)', tweet_text)

# Words with $
cashtags = re.findall(r'\$([A-Za-z][A-Za-z0-9_]*)', tweet_text)

# Unify hashtag and cashtag, excluding duplicates
hashtags = list(set(hashtags + cashtags))

# Merge reply and mentions
if reply and reply not in mentions:
    mentions.insert(0, reply)

# Extract date
date_elem = tweet_elem.find_element(By.CLASS_NAME, "tweet-date")
date_link = date_elem.find_element(By.TAG_NAME, "a")
date_str = date_link.get_attribute('title')
tweet_date = datetime.strptime(date_str, '%b %d, %Y · %I:%M %p %Z')
```

Dataset description

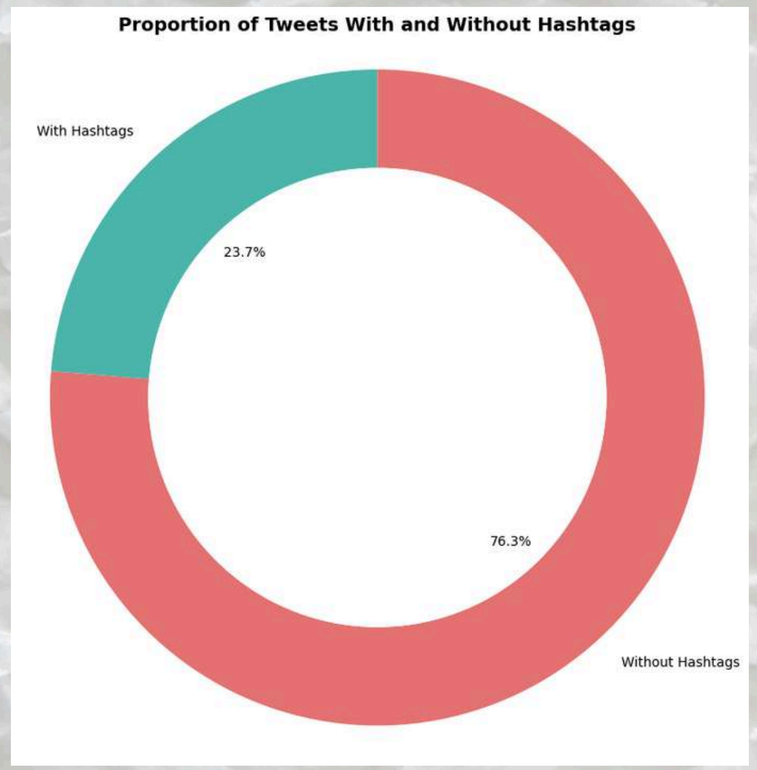
Nitter dataset	
Metric	**Value**
Total number of tweets	10,000
Number of distinct authors	7,104
Average tweets per author	1.41
Total likes	210,277
Total retweets	26,733
Average likes per tweet	21.03
Average retweets per tweet	2.67
Average hashtags per tweet	1.02
Average mentions per tweet	1.19





- 4,155 tweets (41.5%) got ZERO likes
- 8,455 tweets (84.5%) got ZERO retweets

Engagement Statistics



ENGAGEMENT TIERS

- Low engagement (1–5): 4,440 tweets (44.4%)
- No engagement (0 likes/retweets): 4,056 tweets (40.6%)
- Medium engagement (6–50): 1,127 tweets (11.3%)
- High engagement (51–500): 305 tweets (3.0%)
- Viral engagement (>500): 72 tweets (0.7%)

TOP ENGAGED AUTHORS

- Megatron_ron: 19,149 total engagement (16,636 likes, 2,513 retweets)
- Vivek4real_: 15,654 total engagement (13,803 likes, 1,851 retweets)
- theRealKiyosaki: 10,765 total engagement (9,397 likes, 1,368 retweets)

HASHTAG INSIGHTS

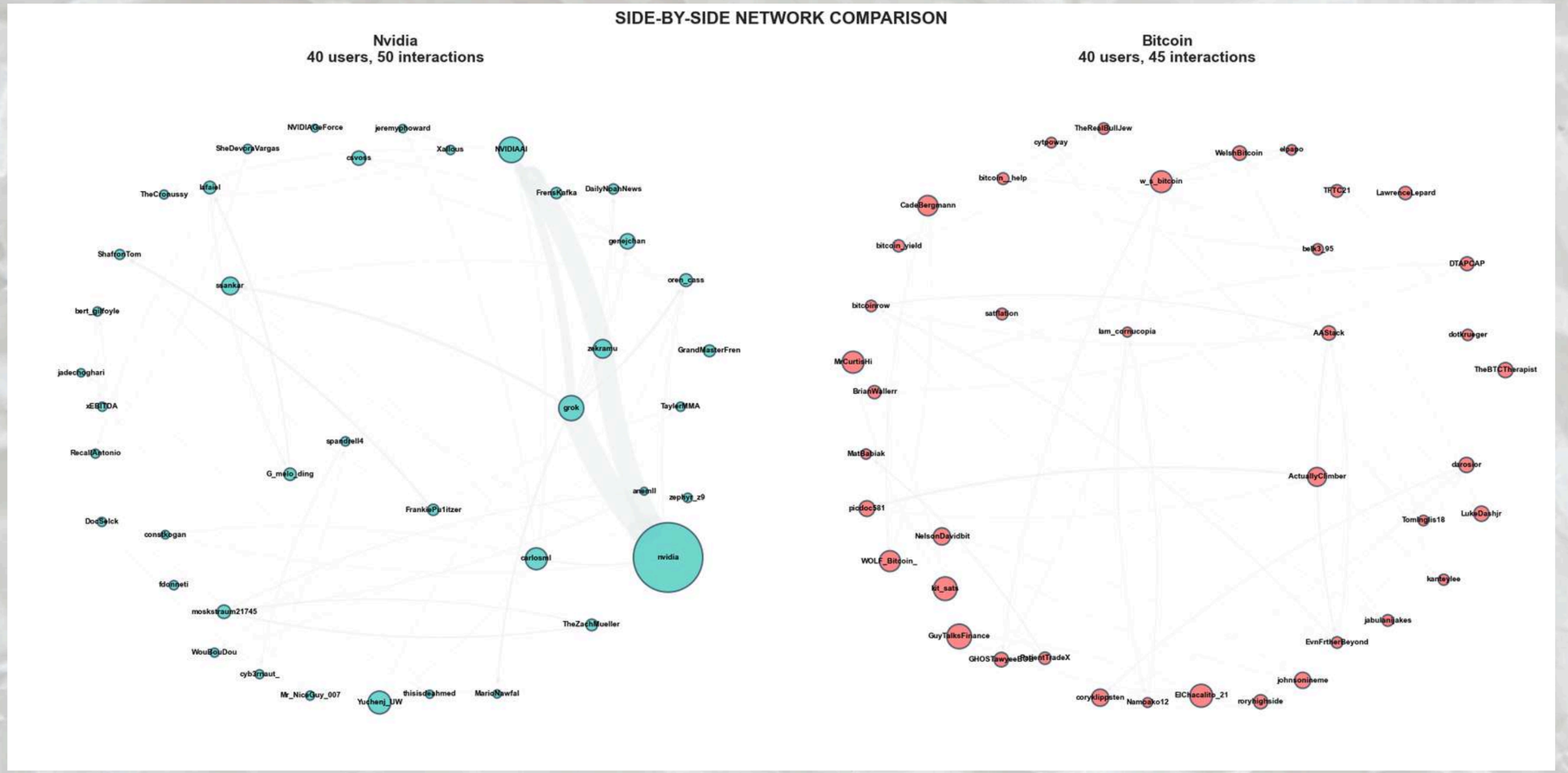
- Tweets with hashtags: 2,372 (23.7%)
- Avg. engagement with hashtags: 27.3
- Avg. engagement without hashtags: 22.6

Twitter Summary statistics Report

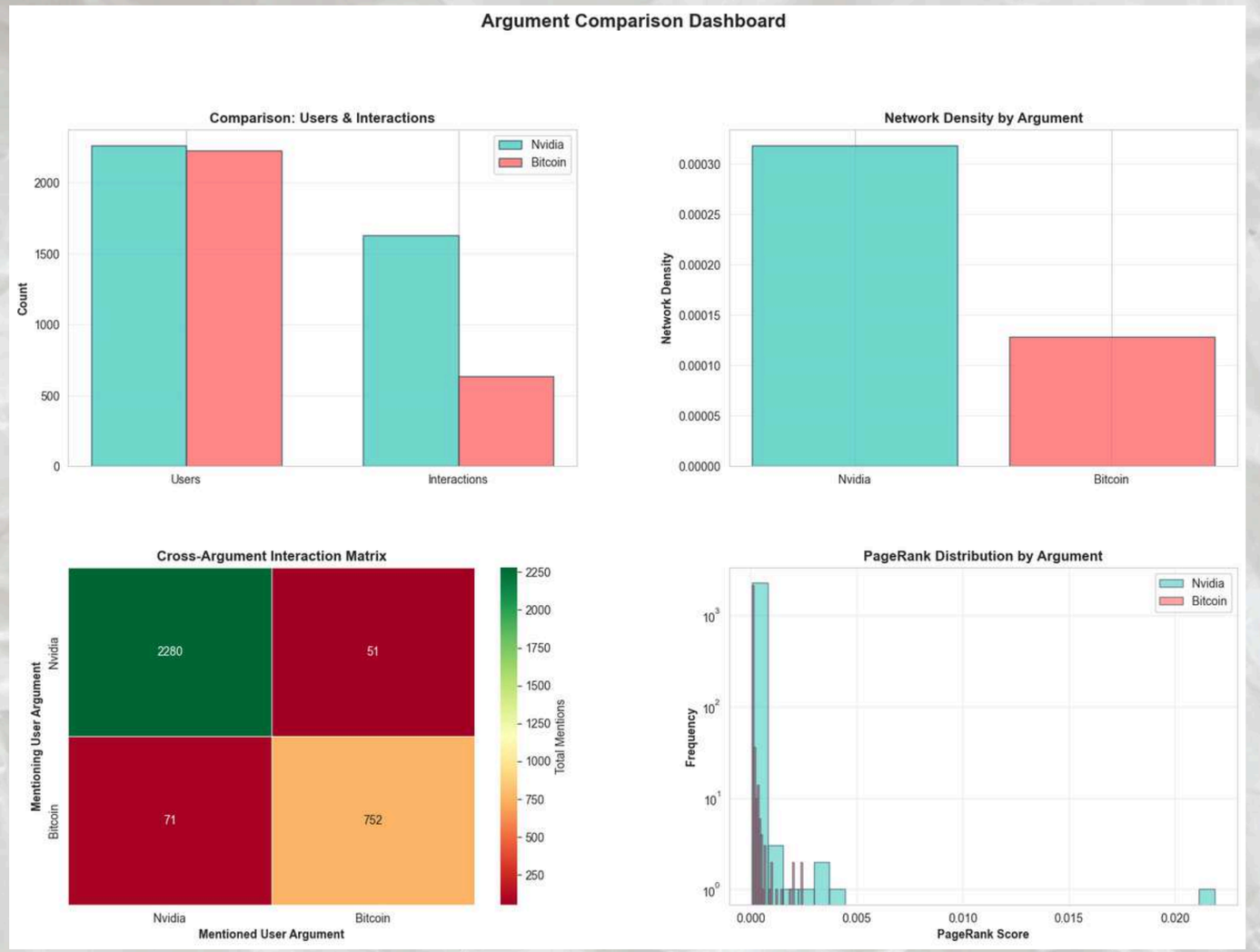
Summary statistics		
Category	**Metric**	**Value**
Dataset Scale	Total Tweets	10000
	Unique Authors	7104
Core Focus	Bitcoin Tweet Share	50%
	Nvidia Tweet Share	50%
Overall Engagement	Average Likes per Tweet	21.03
	Average Retweets per Tweet	2.67
Author Concentration	Most Active Author	grok
	Average Tweets per Author	1.41

404 Tweets
4% of all posts

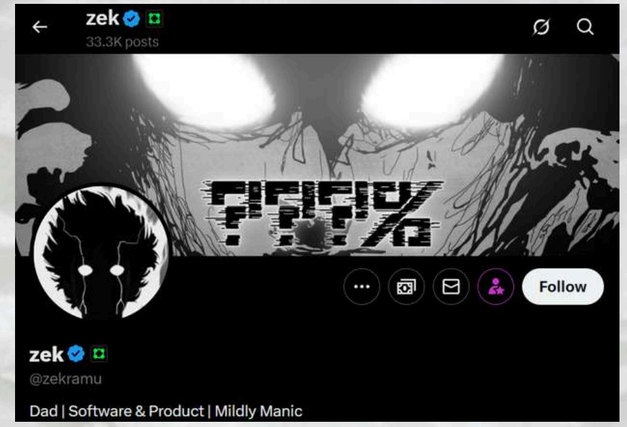
Network Analysis

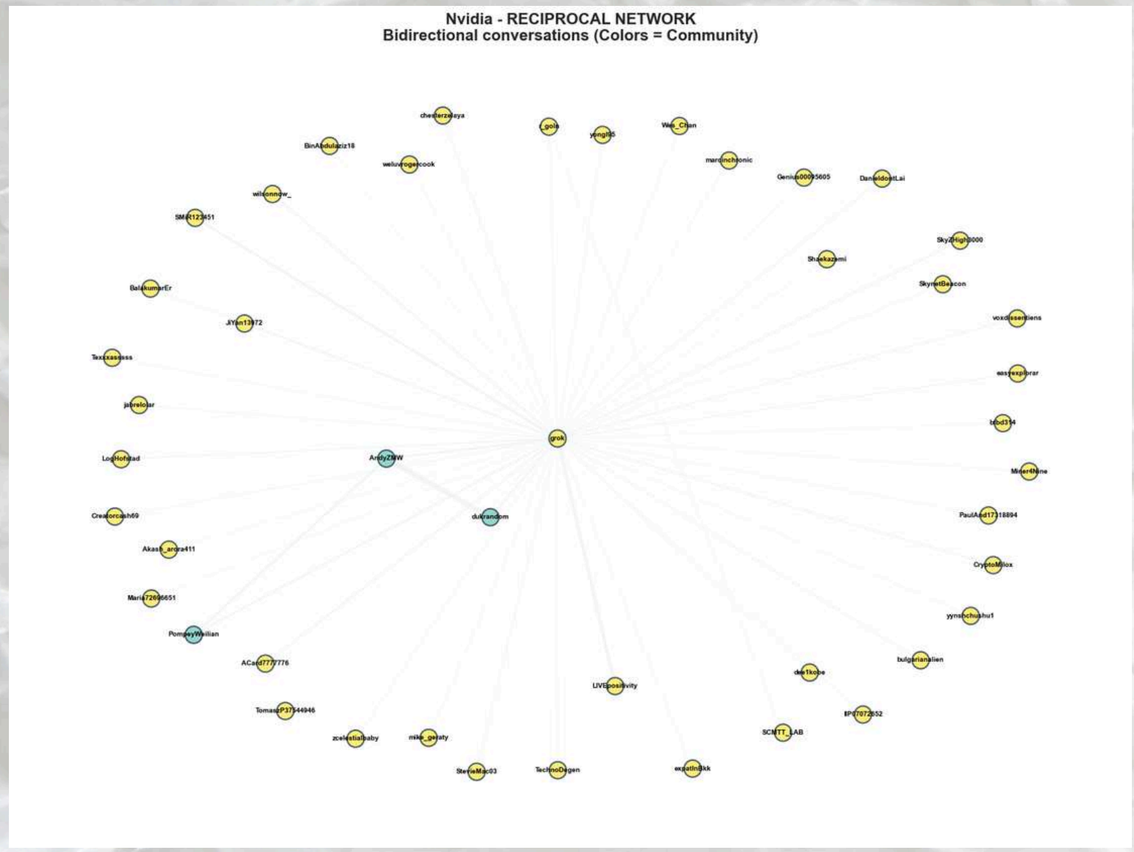


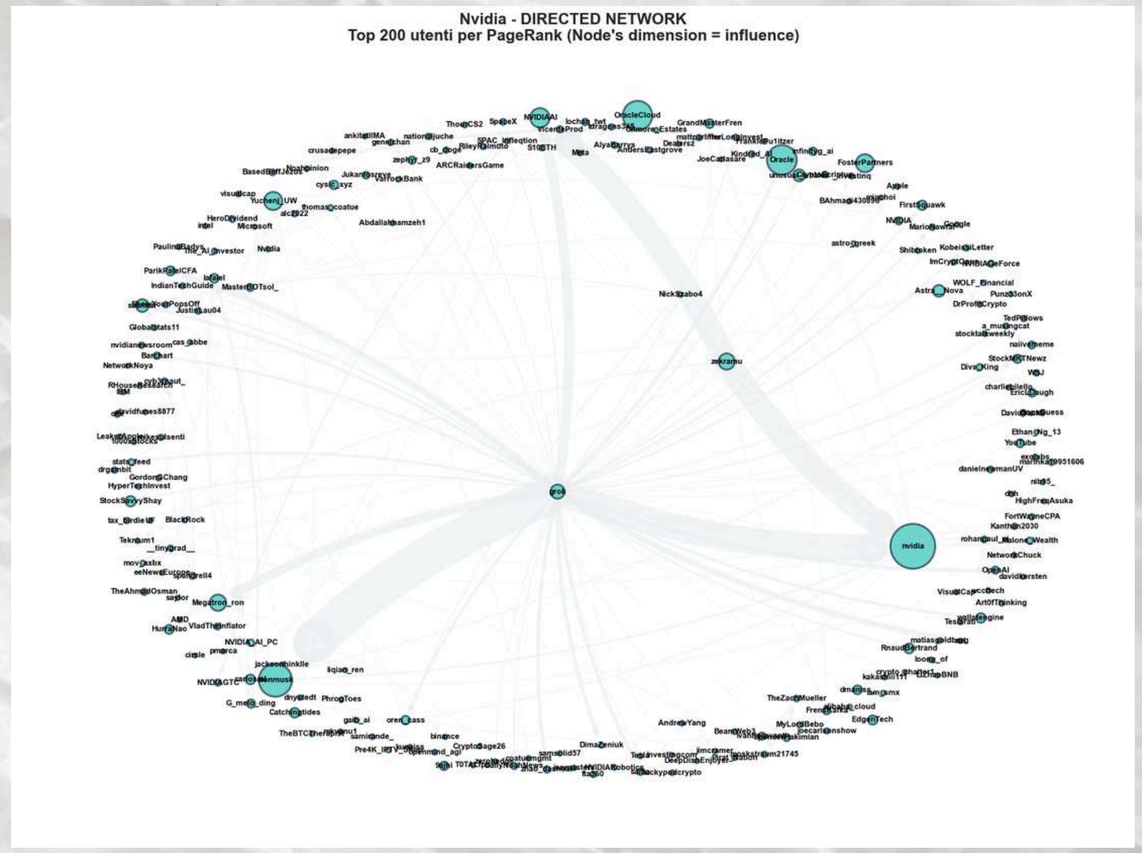
Network Analysis



Network Analysis - Nvidia



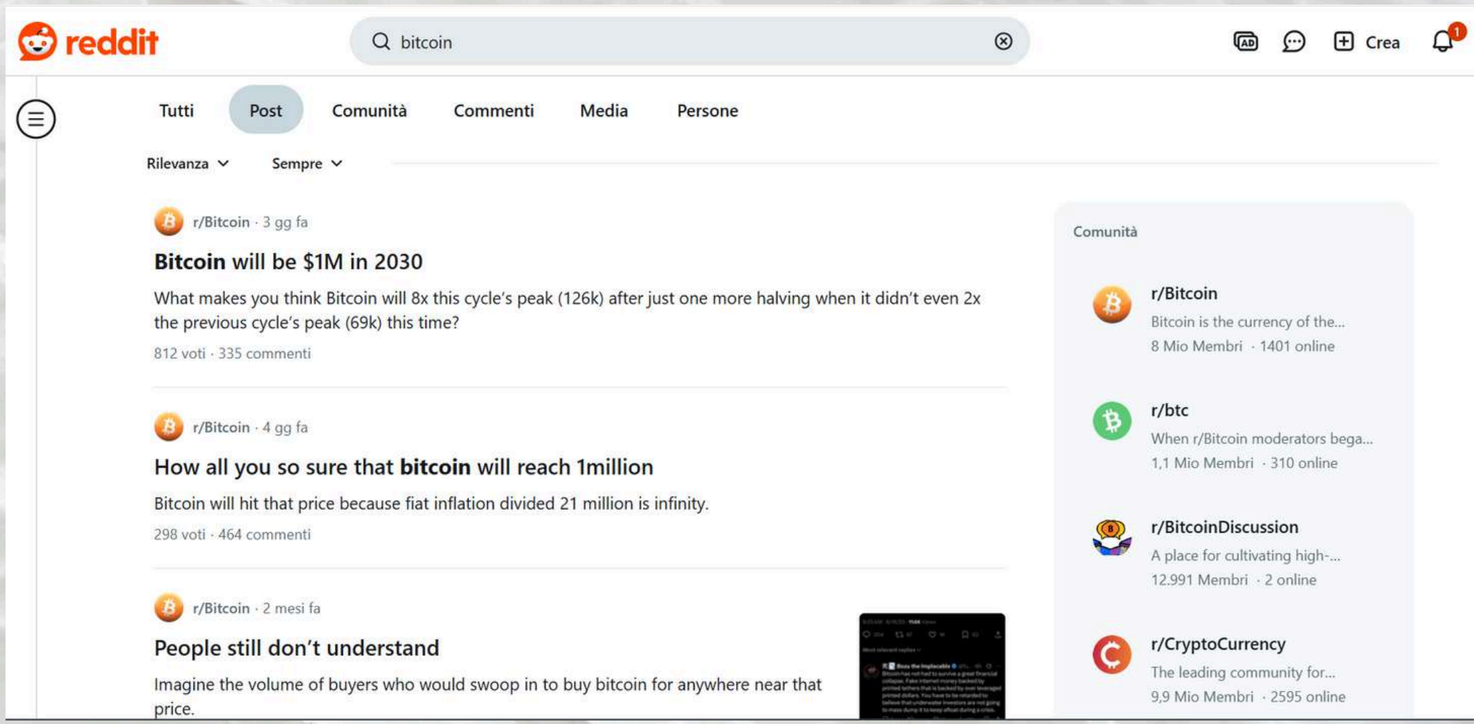




4

REDDIT
Dataset

- Data collection
- Descriptive statistics
- Network Analysis



Data collection

The data collection was conducted using **REDDIT** API among two different: Bitcoin, Nvidia

Scraping procedure

Threads Approach 1

```
for submission in reddit.subreddit(subreddits).new(limit=None):
    title_and_text = (submission.title + " " + (submission.selftext or "")).lower()
    if not any(keyword in title_and_text for keyword in keywords_filter):
        continue
```

Threads Approach 2

```
for submission in reddit.subreddit(subreddits).search(
    "bitcoin OR btc",
    sort="new",
    time_filter="month",
    limit=None
):

    title_and_text = (submission.title + " " + (submission.selftext or "")).lower()
    if not any(keyword in title_and_text for keyword in keywords_filter):
        continue
```

Comments selection

Then download only the comments from 17 to 20 october and filtering them using lang detect library

Reddit Dataset description

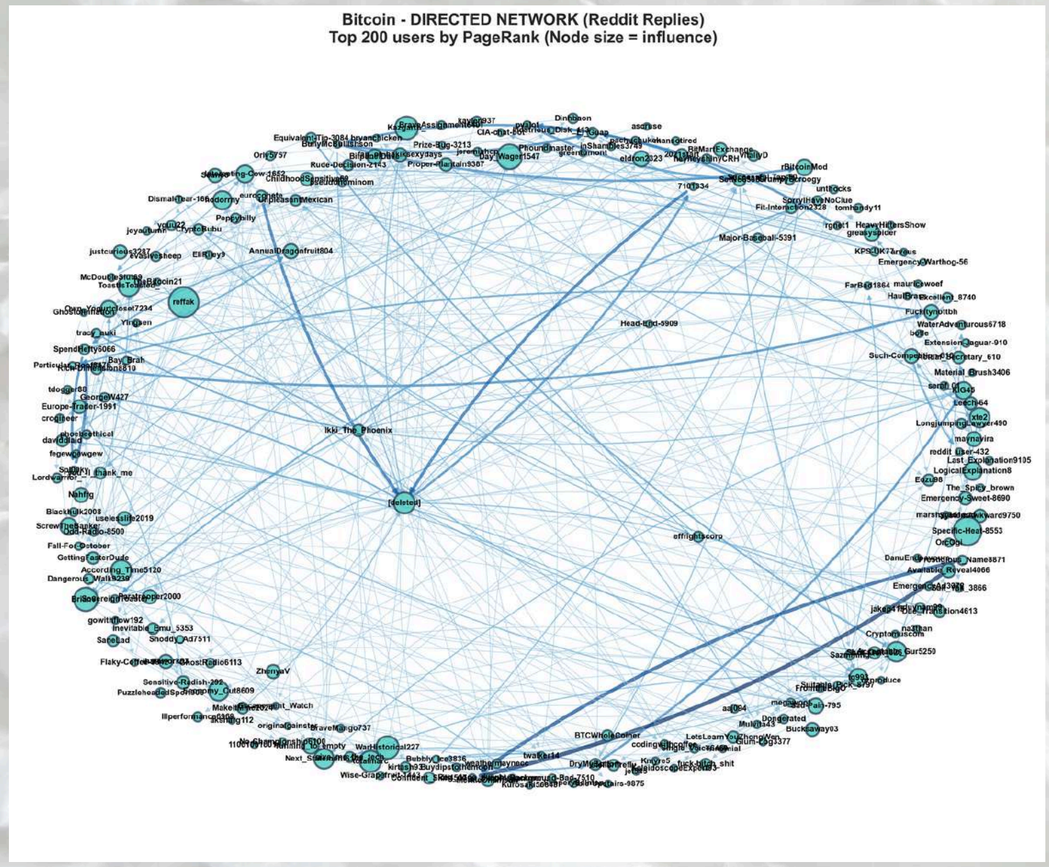
Category	Metric	Value	Interpretation
Core Focus	Top Topic: Bitcoin	80.0% (8,564)	Dominant topic driving most discussions
Core Focus	Secondary Topic: Nvidia	20.0% (2,144)	Secondary but notable discussion topic
Author Activity	Most Active Comment Author	ScrewTheBanker (66)	Top contributor, responsible for 0.6% of comments
Author Activity	Most Active Thread Author	joe4942 (897)	Most prolific thread creator
Author Activity	Average Comments per Author	1.84	Typical engagement per user
Engagement	Total Thread Score	42422	Combined upvote count across all threads
Engagement	Max Thread Score	4603	Highest individual thread upvote score
Engagement	Max Comments in a Thread	789	Peak comment volume in a single discussion
Engagement Distribution	Top 10 Commenters' Share	0.033	Percentage of comments from top 10 contributors

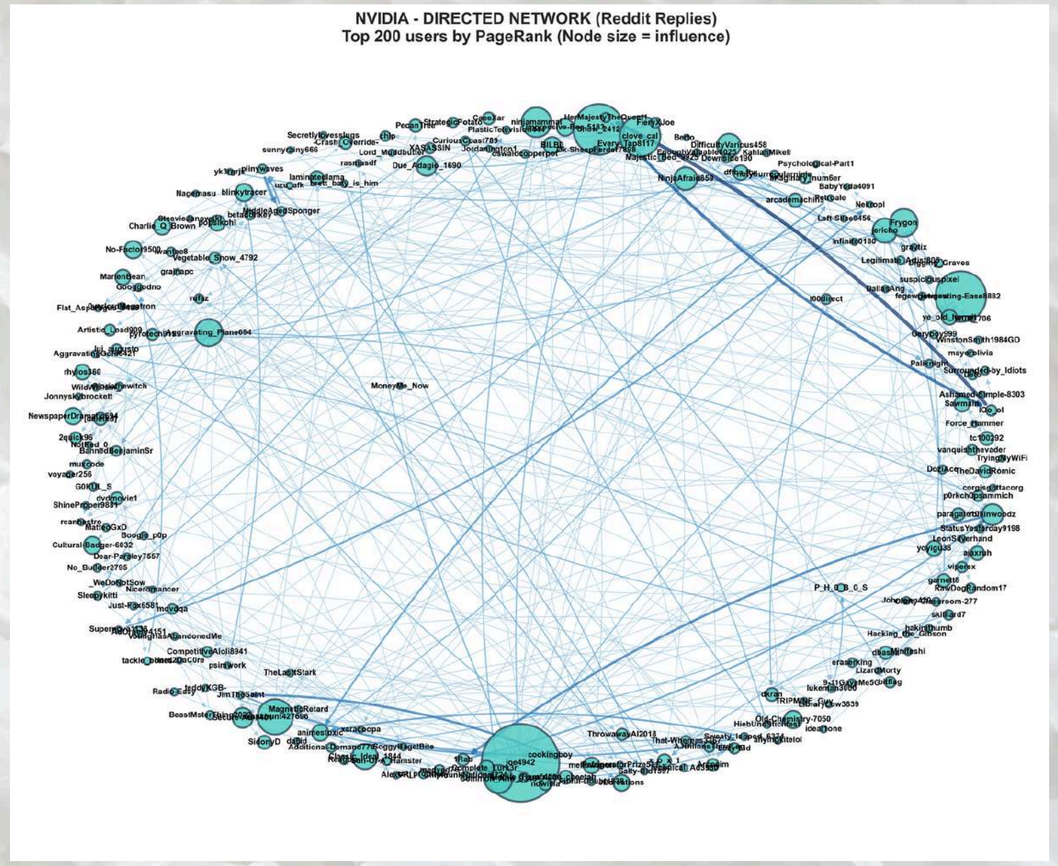
Statistic	Value
Total number of comments	10708
Number of distinct comment authors	5819
Number of distinct threads	335
Number of distinct thread authors	286
Average comments per thread	31.96
Total comment likes	81441
Average likes per comment	7.61
Average thread score	724.7
Average comments per thread (from thread_num_comments)	156

Column Name	Data Type
thread_title	object
thread_author	object
thread_score	int64
thread_num_comments	int64
text_id	object
comment_parent_id	object
text_author	object
text	object
likes	int64
text_date	datetime64[ns]
text_num_replies	int64
retweets	float64
comment_parent_author	object
text_mentions	object
text_hashtags	object
argument	object
site	object
comment_date	datetime64[ns]
hour	int64

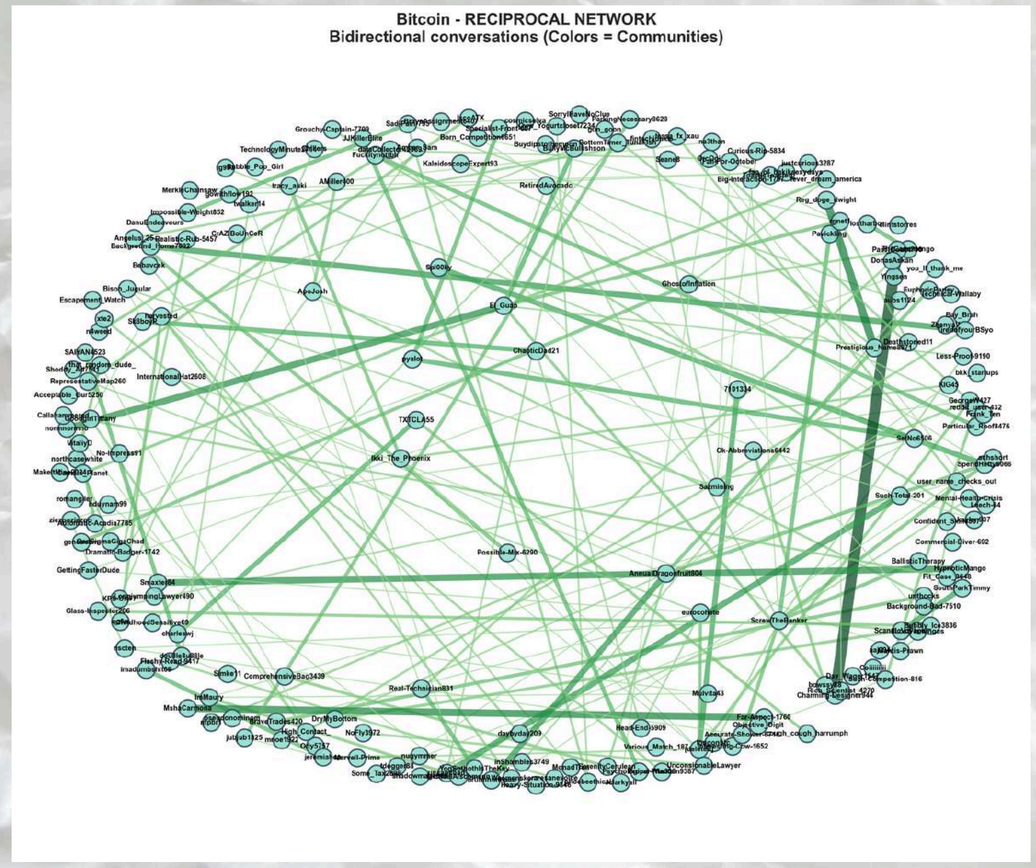
Network Analysis - Directed Network

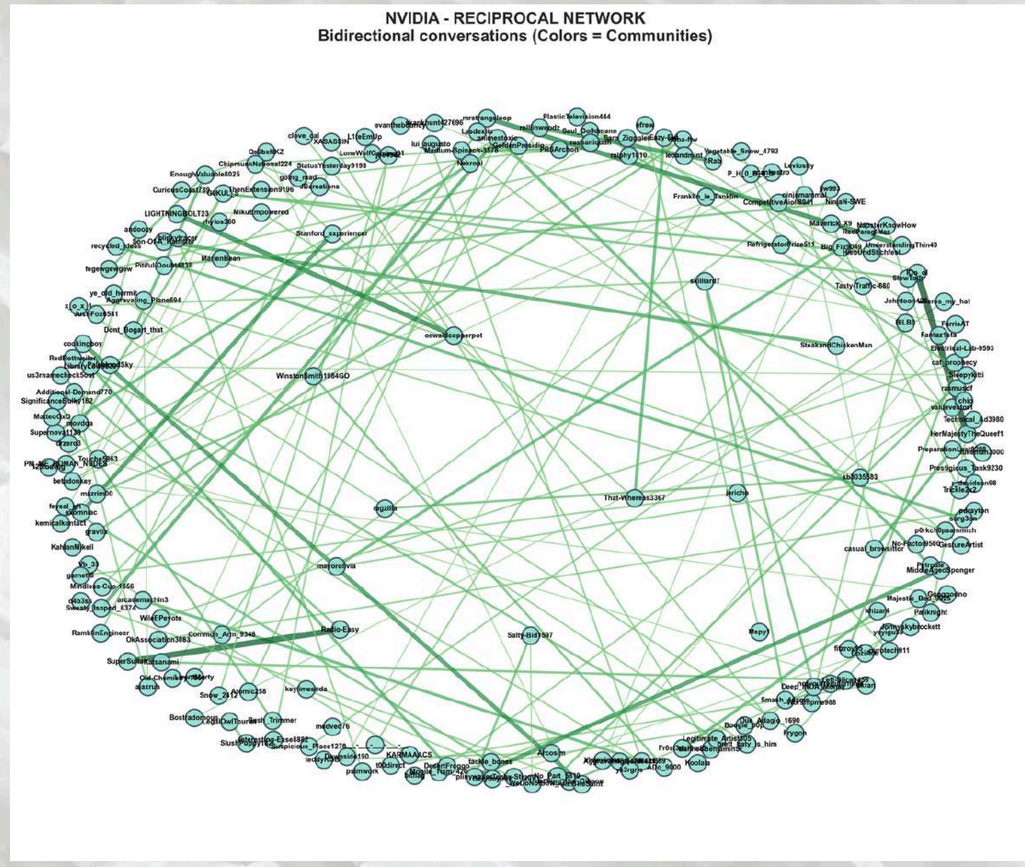
Libraries: Networkx, Matplot



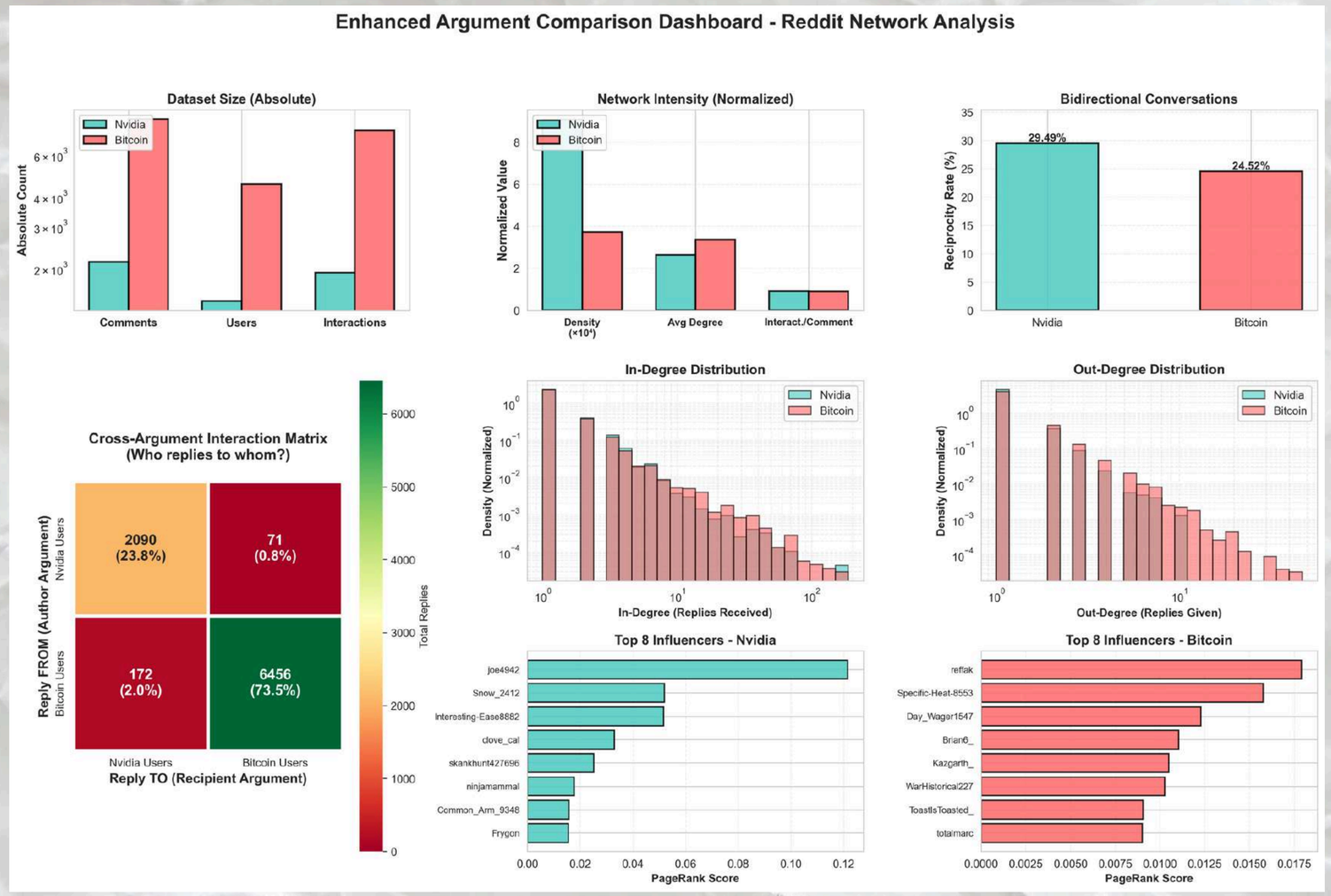


Network Analysis - Reciprocal Network





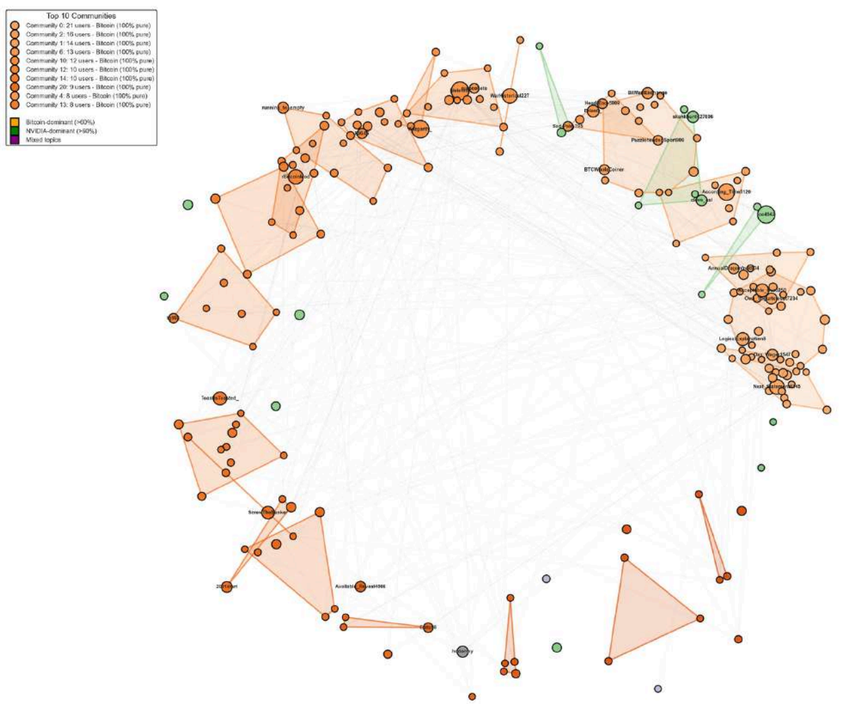
Network Analysis - Comparison



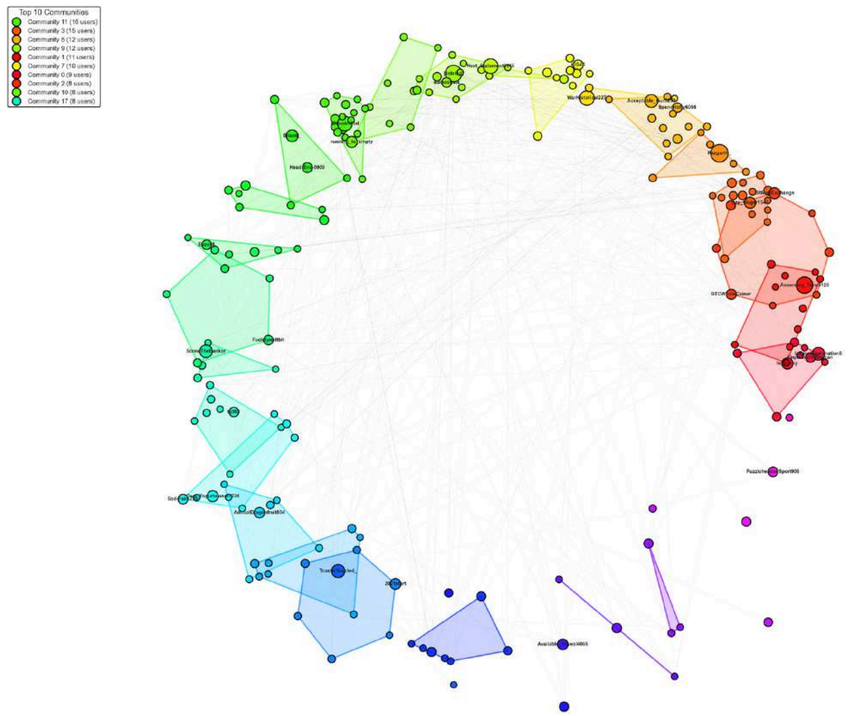
Community detection

Community created with Leiden algorithm.
Resolution: 1.0. Partition: RBConfigurationVertexPartition. Library: Leidenalg

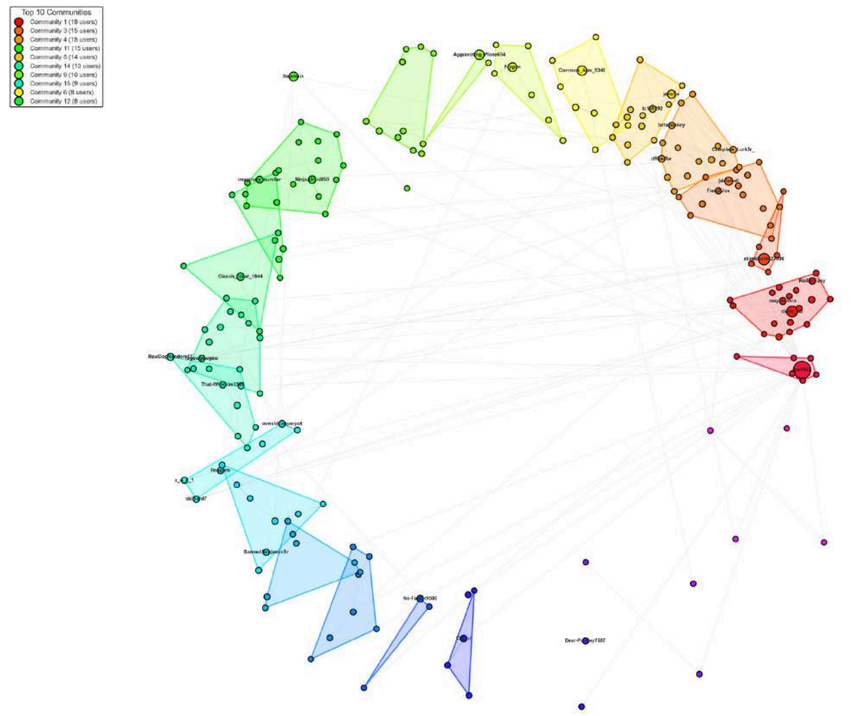
Bitcoin and Nvidia



Bitcoin



Nvidia

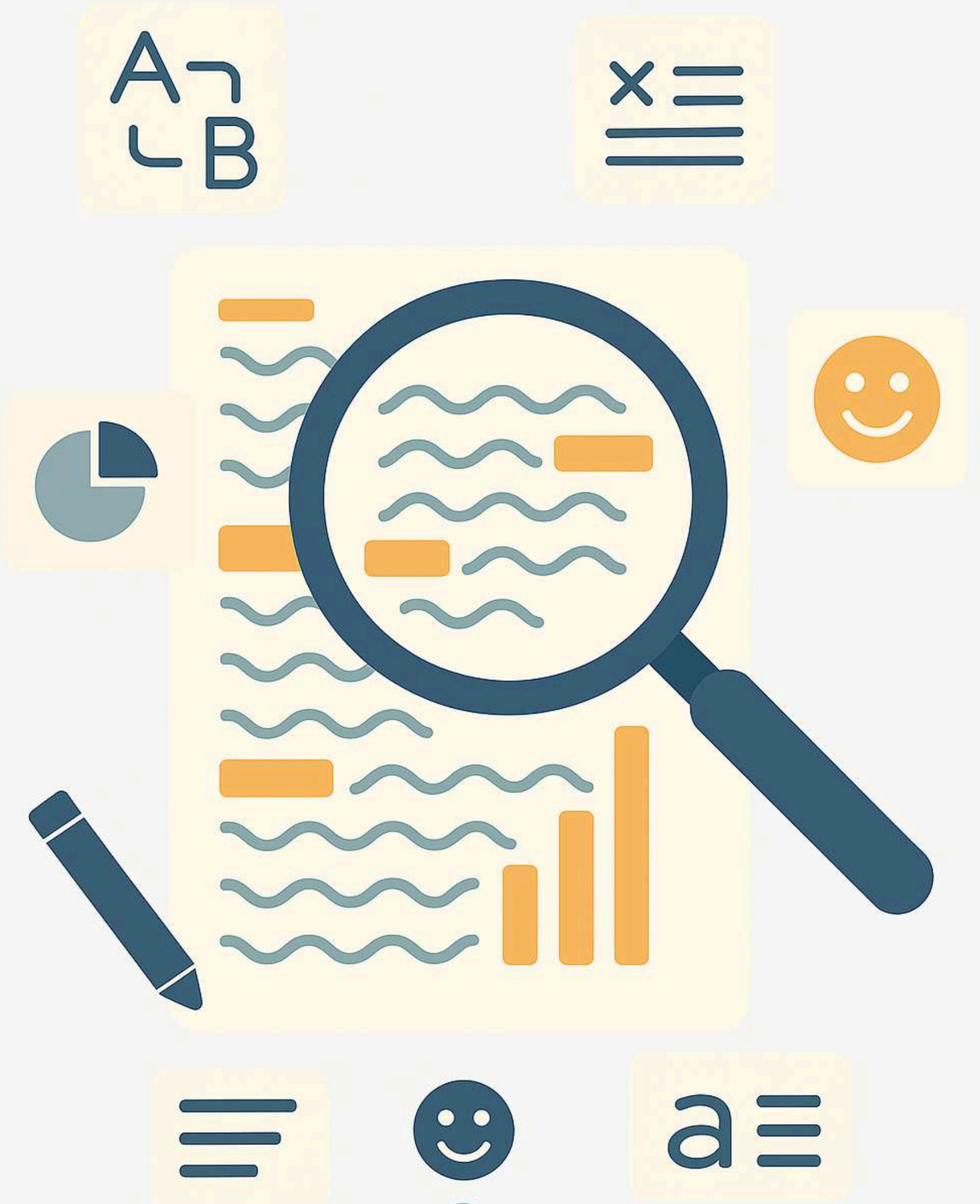


5

NITTER & REDDIT

Pre-processing

- Text cleaning
- Multi-words detection
- Collocations
- Lemmatization
- Removing stopwords
- Vectorization



Libraries:

- **re (Regular Expressions)**
  - Used to remove URLs, mentions, special symbols (#, $), HTML entities, and extra spaces.
  - Enables powerful pattern matching with compact expressions
- **spaCy**
  - Core NLP framework for linguistic analysis.
  - Performs POS tagging, lemmatization, and syntactic parsing to detect collocations and multi-word expressions.
  - Efficient and accurate thanks to pre-trained models
- **NLTK (Natural Language Toolkit)**
  - Provides built-in stopword lists for multiple languages.
  - Used to remove non-informative words
- **sklearn (scikit-learn)**
  - Machine learning library used for text vectorization.
  - CountVectorizer converts text into a Document-Term Matrix, turning words into numerical frequency vectors.

Text cleaning:

```
# Remove URLs
text = re.sub(r"(f|ht)(tp)(s?)(://)(.*?)([\s]|$)", " ", text)

# Remove RT patterns
text = re.sub(r"(RT|rt|via)((?:\b\W*@\w+)+)", " ", text)

# Remove HTML entities
html_entities = ["&copy;", "&reg;", "&trade;", "&ldquo;", "&lsquo;", "&rsquo;",
                 "&bull;", "&middot;", "&ndash;", "&mdash;", " ", "&lt;",
                 "&gt;", "&amp;", "&quot;"]
for entity in html_entities:
    text = text.replace(entity, " ")

# Remove mentions (@username)
text = re.sub(r"@\S+", " ", text)

# Remove emojis (including flag emojis like IN)
if remove_emojis:
    # Remove emoji characters (comprehensive pattern)
    text = re.sub(r"[^\w\s,.\'!?-]", "", text)
```

```
# Handle $ symbol:
# Remove $ only before letters (not before numbers)
if remove_cashtag_before_words:
    text = re.sub(r'\$(?=[A-Za-z])', '', text)

# Handle # symbol:
# Remove '#' but keep the word after it
if remove_hashtag_symbol:
    text = re.sub(r'#(\w+)', r'\1', text)

# Normalize whitespace
text = re.sub(r"\s+", " ", text).strip()

# Lowercase if requested
if lowercase:
    text = text.lower()

return text
```

Collocations & Multi-words detection:

```python
def extract_collocations_POS(texts, nlp, pos_patterns=[('ADJ','NOUN'), ('NOUN','NOUN'),
                                                       ('NOUN','PROPN'), ('PROPN','PROPN')],
                             min_freq=2, save_file="colloc_POS.xlsx", verbose=True):

    annotated = annotate_texts(nlp, texts, show_progress=verbose)
    counts = Counter()
    pattern_map = {}

    for doc in annotated:
        for i in range(len(doc)-1):
            t1, t2 = doc[i], doc[i+1]
            pattern = (t1['upos'], t2['upos'])

            if pattern in pos_patterns:
                w1 = (t1['lemma'] if t1['lemma'] != '_' else t1['form']).lower()
                w2 = (t2['lemma'] if t2['lemma'] != '_' else t2['form']).lower()
                colloc = f"{w1} {w2}"
                counts[colloc] += 1
                pattern_map[colloc] = f"{pattern[0]} {pattern[1]}"
```

```python
for colloc in collocations:
    escaped = r'\s+'.join(re.escape(word) for word in colloc.split())
    pattern = re.compile(rf'\b{escaped}\b', flags=re.IGNORECASE)
    replacement = "_".join(colloc.split())
    patterns.append((pattern, replacement))
```

```python
def extract_collocations_PMI(texts, nlp, top_n=200, min_freq=2,
                             save_file="colloc_PMI.xlsx", verbose=True):

    annotated = annotate_texts(nlp, texts, show_progress=verbose)
    unigram = Counter()
    bigram = Counter()
    total_unigrams = 0

    for doc in annotated:
        lemmas = []
        for tok in doc:
            if tok['upos'] == 'PUNCT':
                continue
            lemma = (tok['lemma'] if tok['lemma'] != '_' else tok['form']).lower()
            lemmas.append(lemma)
            unigram[lemma] += 1
            total_unigrams += 1

        for i in range(len(lemmas)-1):
            bigram[f"{lemmas[i]} {lemmas[i+1]}"] += 1

    N = max(total_unigrams, 1)
    rows = []

    for big, freq in bigram.items():
        if freq < min_freq:
            continue

        w1, w2 = big.split(" ", 1)
        p_w1 = unigram[w1] / N
        p_w2 = unigram[w2] / N
        p_w1w2 = freq / max(1, N-1)

        if p_w1 > 0 and p_w2 > 0 and p_w1w2 > 0:
            pmi = math.log2(p_w1w2 / (p_w1 * p_w2))
        else:
            pmi = float('-inf')
```

Lemmatization & Stopwords

```python
# Download NLTK stopwords
try:
    nltk_stopwords.words('english')
```

```python
def create_text_nostop(df, lemmatized_df, verbose=True):
    """Create text_nostop column WITHOUT stopwords (content words only)"""

    if verbose:
        print("\nCreating text_nostop column (without stopwords)...")

    def reconstruct_nostop(doc_idx):
        doc_id = f"doc_{doc_idx}"
        # Filter only content words (STOP == False)
        content_lemmas = lemmatized_df[
            (lemmatized_df['doc_id'] == doc_id) &
            (~lemmatized_df['STOP'])
        ]['lemma'].tolist()
        return ' '.join(content_lemmas)

    df['text_nostop'] = [reconstruct_nostop(i) for i in range(len(df))]

    if verbose:
        avg_tokens = df['text_nostop'].str.split().str.len().mean()
        print(f"✓ Created text_nostop column")
        print(f"  Avg tokens: {avg_tokens:.1f}")

    return df
```

```python
def lemmatize_texts(texts, nlp=None, stopwords_list=None, verbose=True):
    """Lemmatize texts and identify stopwords"""

    if nlp is None:
        nlp = load_spacy_model()

    if stopwords_list is None:
        stopwords_list = list(nltk_stopwords.words('english'))

    stopwords_lower = set(w.lower() for w in stopwords_list)

    results = []
    iterator = tqdm(enumerate(texts), total=len(texts), desc="Lemmatizing") if verbose else enumerate(texts)

    for doc_idx, text in iterator:
        doc = nlp(str(text) if pd.notna(text) else "")
        doc_id = f"doc_{doc_idx}"

        for token_idx, token in enumerate(doc):
            if token.is_punct or token.is_space:
                continue

            is_stopword = (token.text.lower() in stopwords_lower or
                           token.lemma_.lower() in stopwords_lower)

            results.append({
                'doc_id': doc_id,
                'token_id': token_idx + 1,
                'token': token.text,
                'lemma': token.lemma_,
                'upos': token.pos_,
                'STOP': is_stopword
            })

    df_lem = pd.DataFrame(results)

    if verbose:
        print(f"\n✓ Lemmatization complete: {len(df_lem)} tokens from {len(texts)} documents")
        print(f"  Stopwords: {df_lem['STOP'].sum()}")
        print(f"  Content words: {(~df_lem['STOP']).sum()}")

    return df_lem
```

Vectorization:

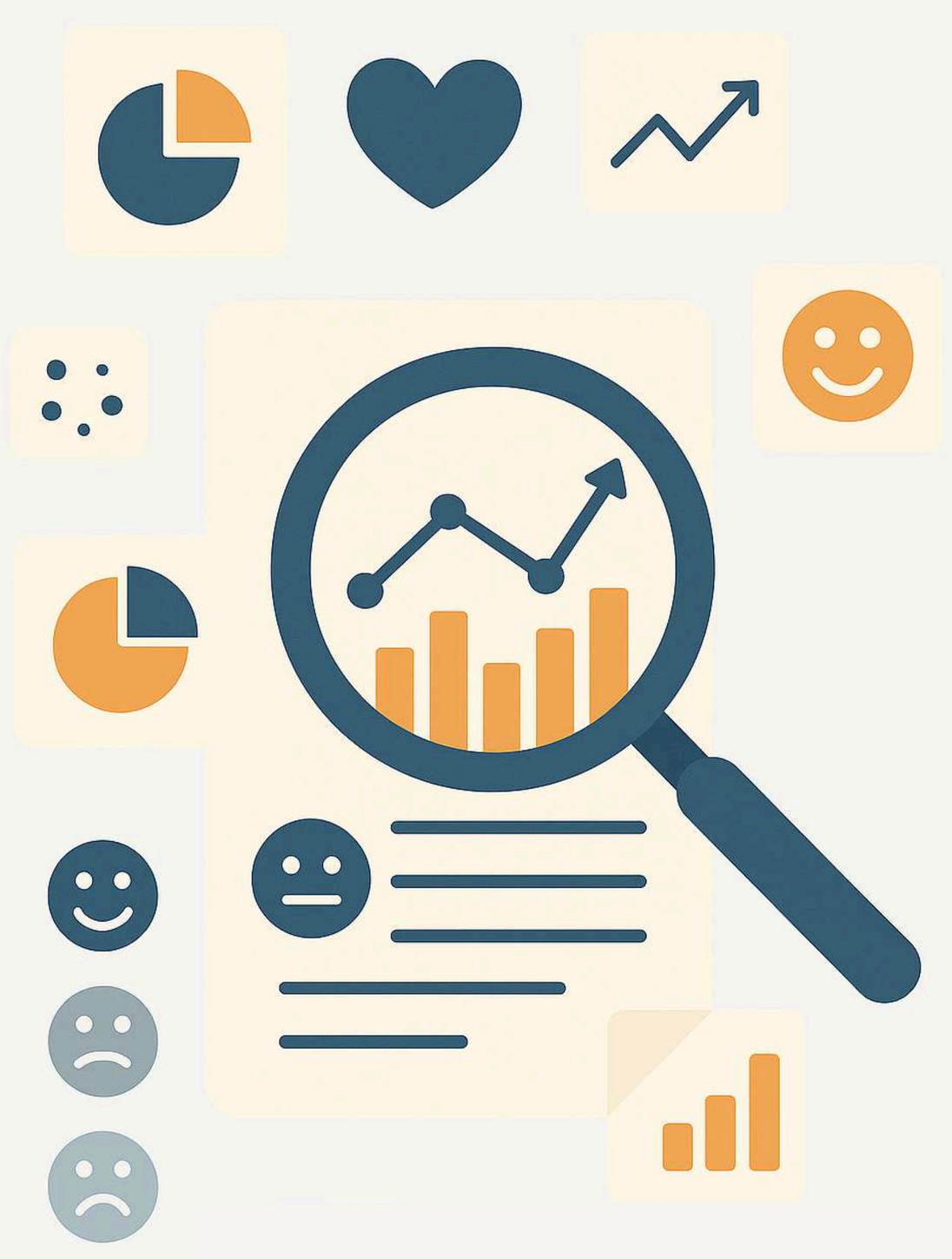
```
def create_document_term_matrix(texts, save_file="document_term_matrix.csv", verbose=True):

    vectorizer = CountVectorizer(
        lowercase=True,
        token_pattern=r'(?u)\b\w+\b',
        min_df=1
    )

    dtm_sparse = vectorizer.fit_transform(texts)

    # Transpose: rows=terms, columns=documents
    dtm_df = pd.DataFrame(
        dtm_sparse.toarray().T,
        index=vectorizer.get_feature_names_out(),
        columns=[f'doc_{i}' for i in range(len(texts))]
    )
```

6

NITTER & REDDIT

Text Analysis

- Explorative data analysis
- Sentiment analysis



(EDA) dataset overview and statistics

Column	Type
thread_title	object
thread_author	object
thread_score	float64
thread_num_comments	float64
text_id	object
comment_parent_id	object
text_author	object
text	object
likes	int64
text_date	object
text_num_replies	int64
retweets	float64
comment_parent_author	object
text_mentions	object
text_hashtags	object
argument	object
site	object
Unnamed: 0	float64
comment_date	datetime64[ns]
hour	float64

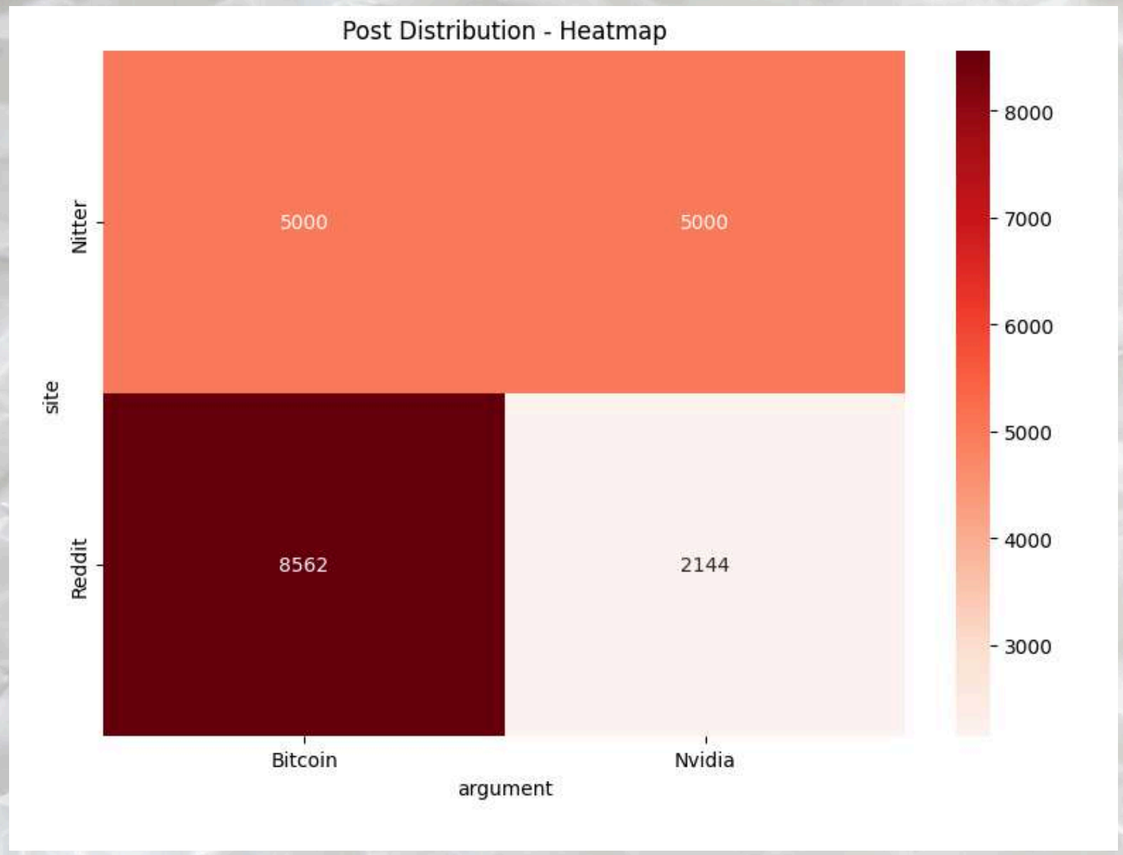
Platform statistics

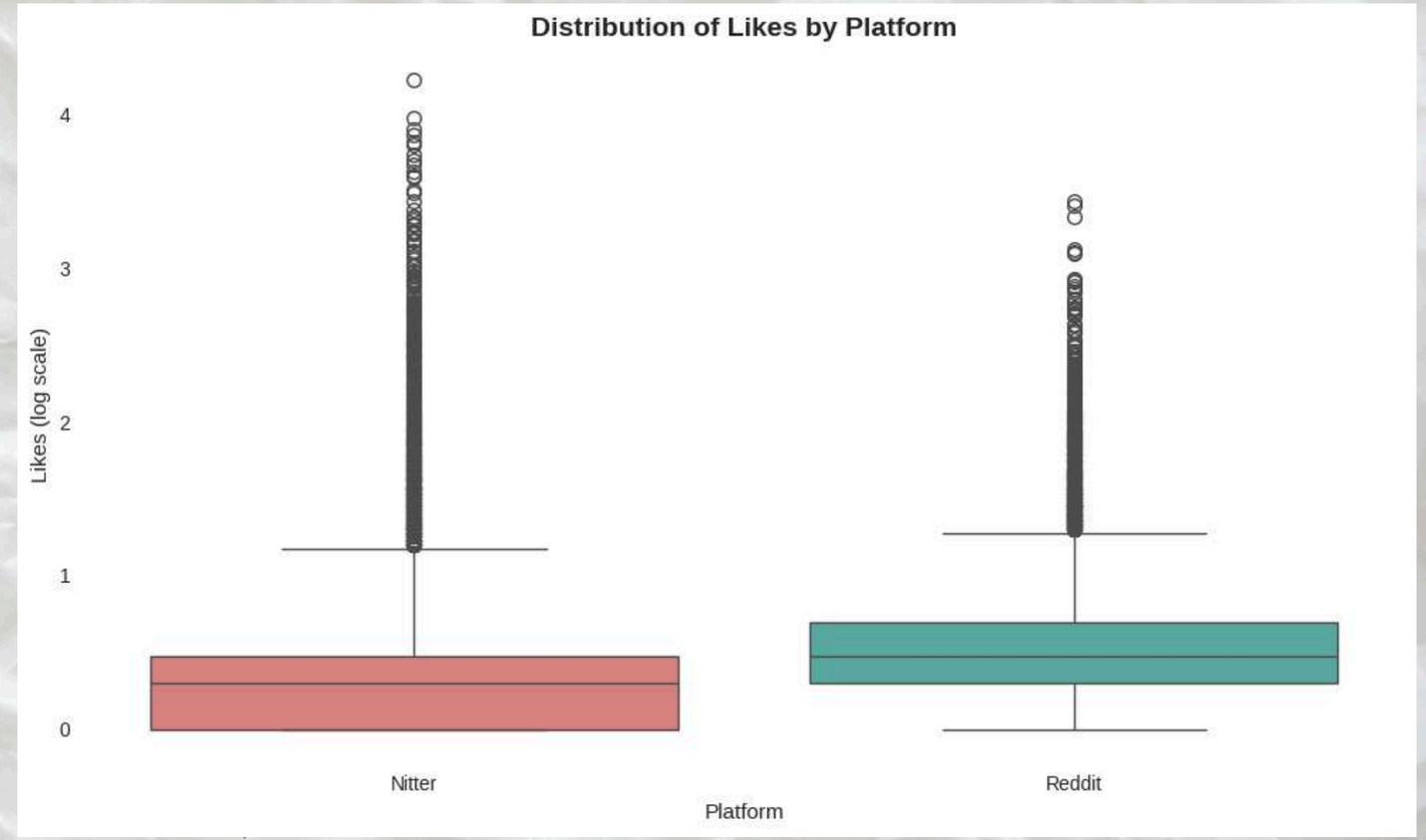
Site	Posts (n)	Avg Likes	Median Likes	Std Likes	Avg Replies	Median Replies	Std Replies	Unique Authors	Posts / Author
Nitter	10,000	21.03	1	281.72	2.08	0	19.56	7,104	1.41
Reddit	10,708	7.61	2	55.83	0.58	0	1.2	5,819	1.84

Ticker Statistics by Platform

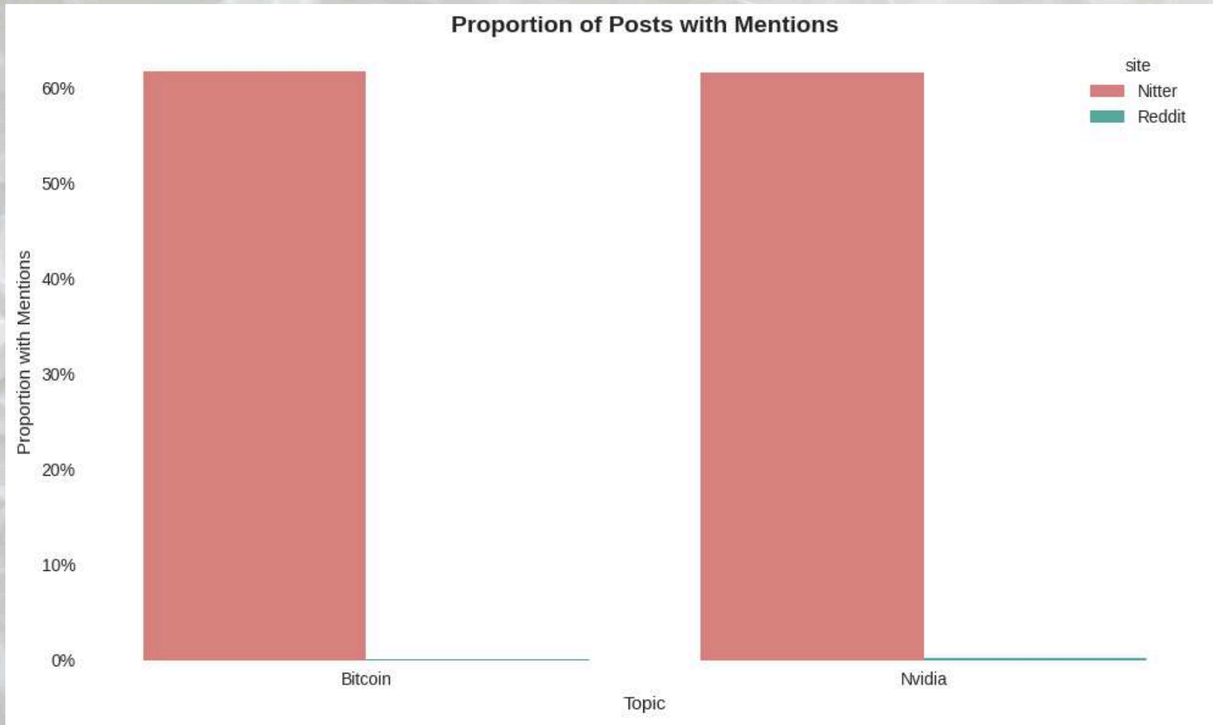
Site	Argument	Posts (n)	Avg Likes	Median Likes	Avg Replies	Median Replies
Nitter	Bitcoin	5,000	17.61	1	2.09	0
Nitter	Nvidia	5,000	24.44	1	2.07	0
Reddit	Bitcoin	8,564	5.16	2	0.54	0
Reddit	Nvidia	2,144	17.37	2	0.76	0

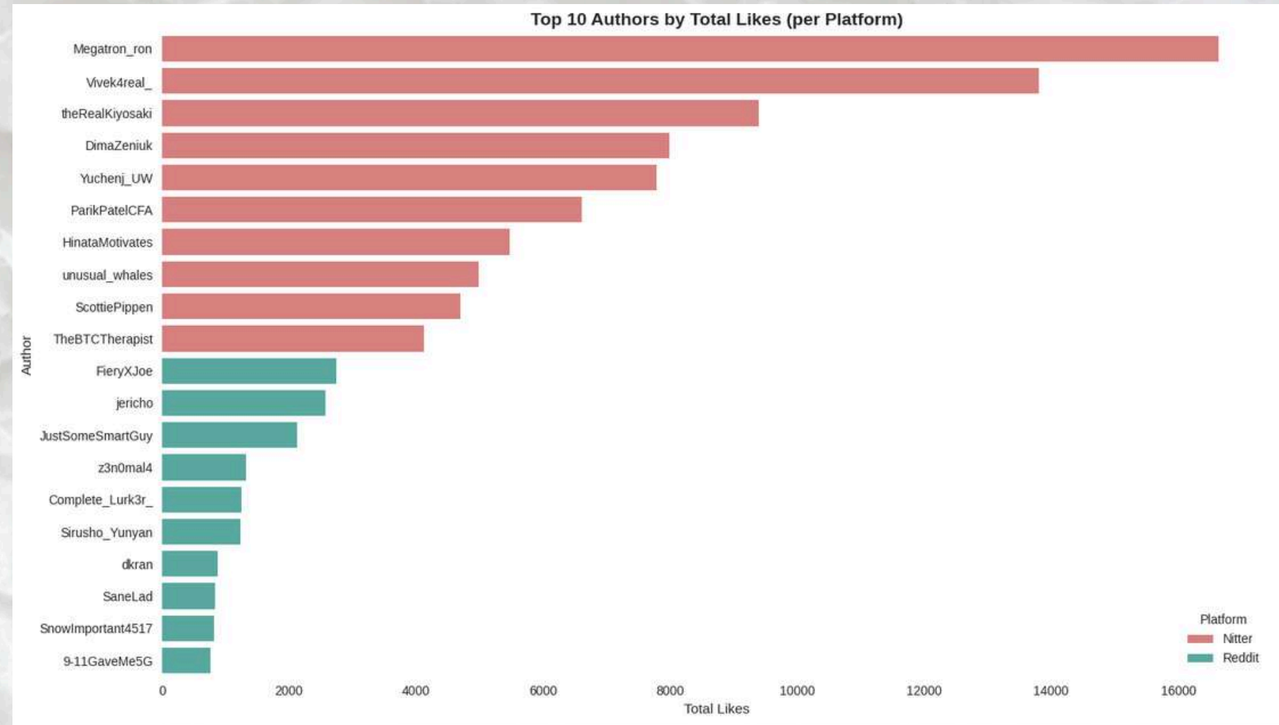
Data Distribution Analysis





Overview | Dataset | Pre-processing | Text analysis





Comparative Word Cloud Analysis Nitter vs Reddit

Sentiment analysis

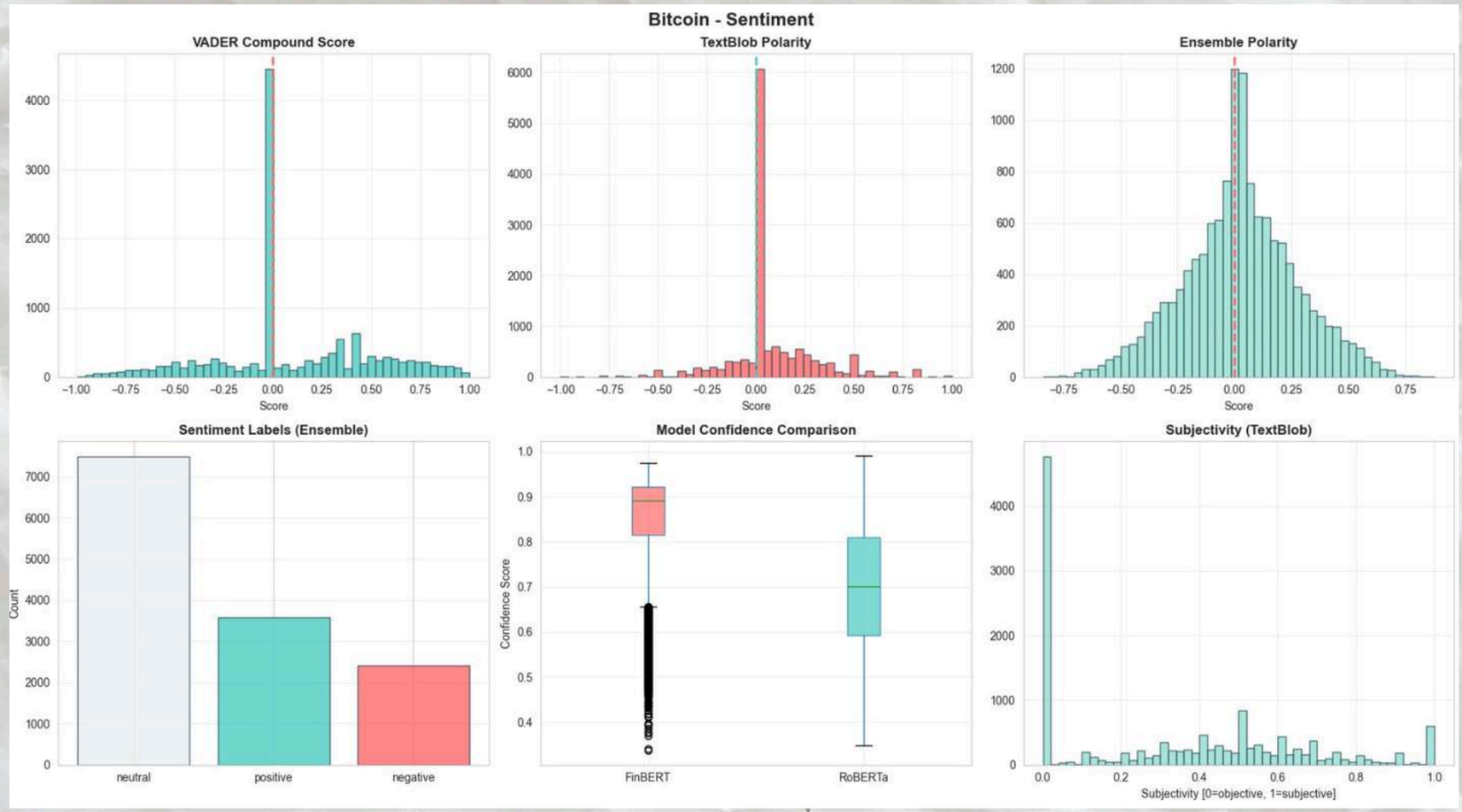
Polarity approaches

Vader → A rule-based model specifically tuned for social media text. It combines lexical features and heuristics to estimate polarity scores. Library: NLTK
TextBlob → A lexicon and rule-based model that uses predefined word sentiment scores and simple NLP techniques to compute text polarity and subjectivity. Library: TextBlob
RoBERTa → A transformer-based language model fine-tuned for sentiment analysis. It captures nuanced contextual meanings to predict polarity with high accuracy. Library: Hugging Face
FinBERT → A RoBERTa-based model fine-tuned on financial text to detect sentiment in news, reports, and market-related documents. Library: Hugging Face

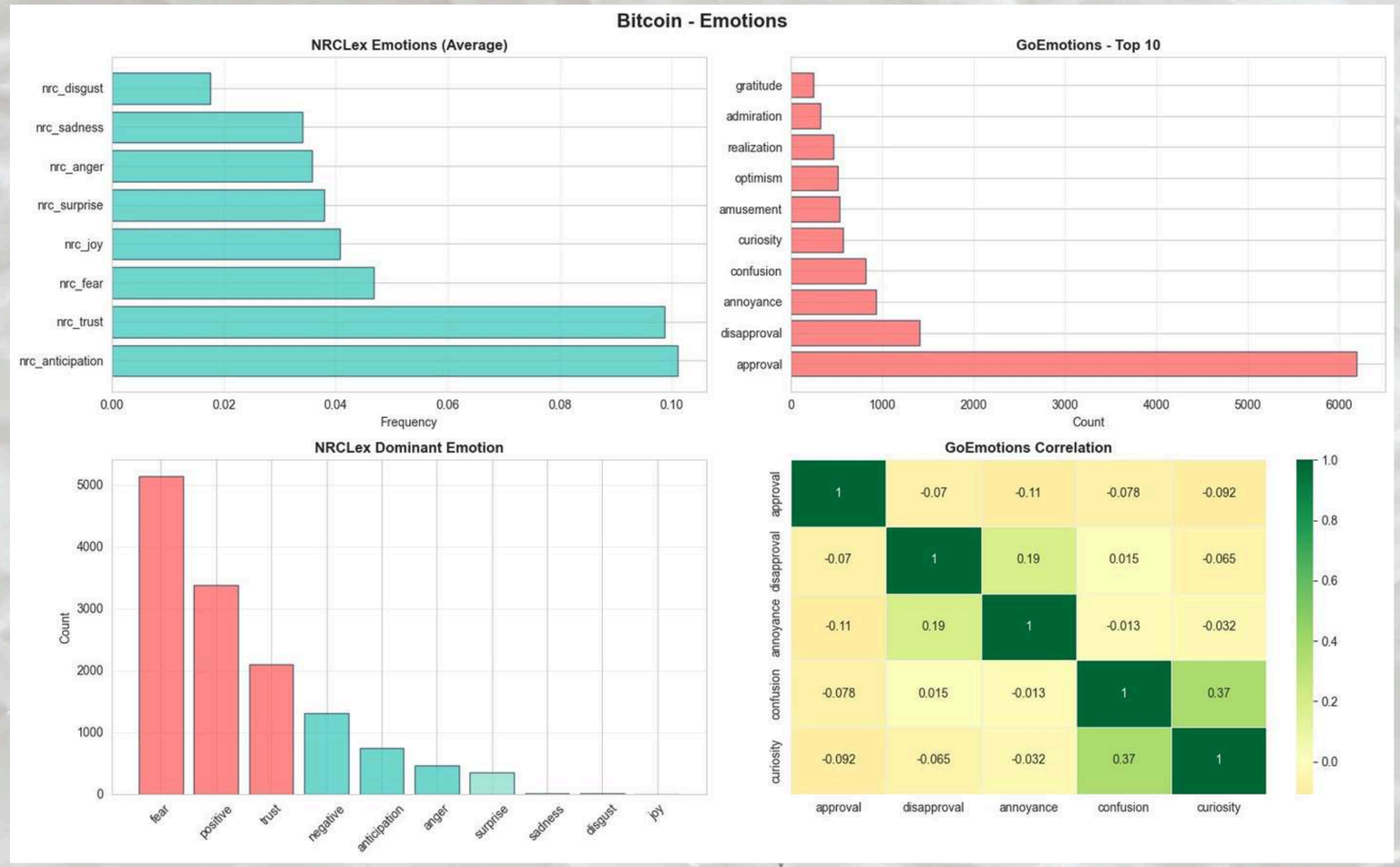
Emotion approaches

NRC → a lexicon-based model that associates words with eight basic emotions and two sentiments: positive/negative. It provides a simple, interpretable way to detect emotional tone in text without machine learning. Library: nrclex
GoEmotions → A BERT-based deep learning model trained on 58k Reddit comments to classify 27 fine-grained emotions + neutral. It's context aware. Library: Hugging Face

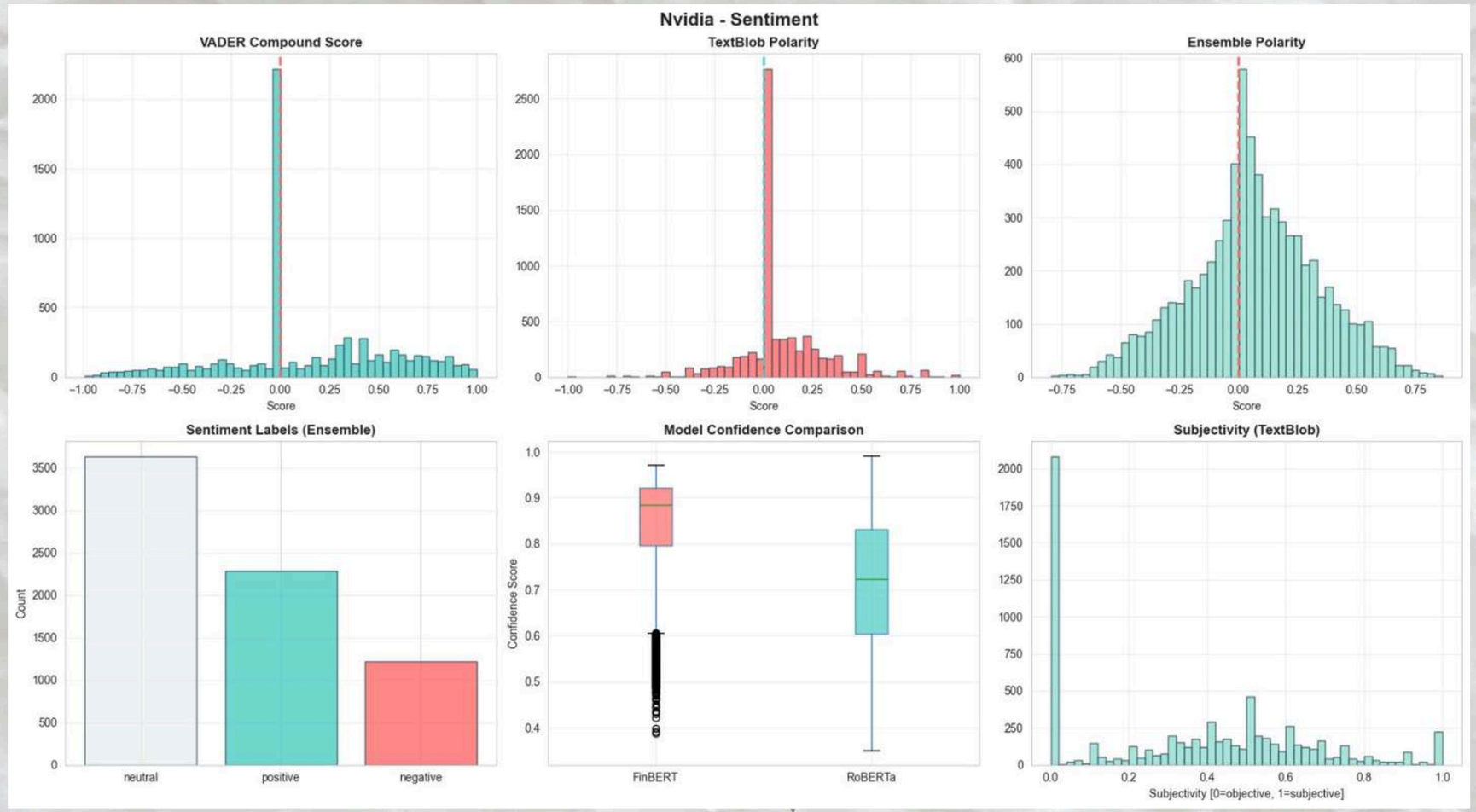
Sentiment analysis: Bitcoin polarity



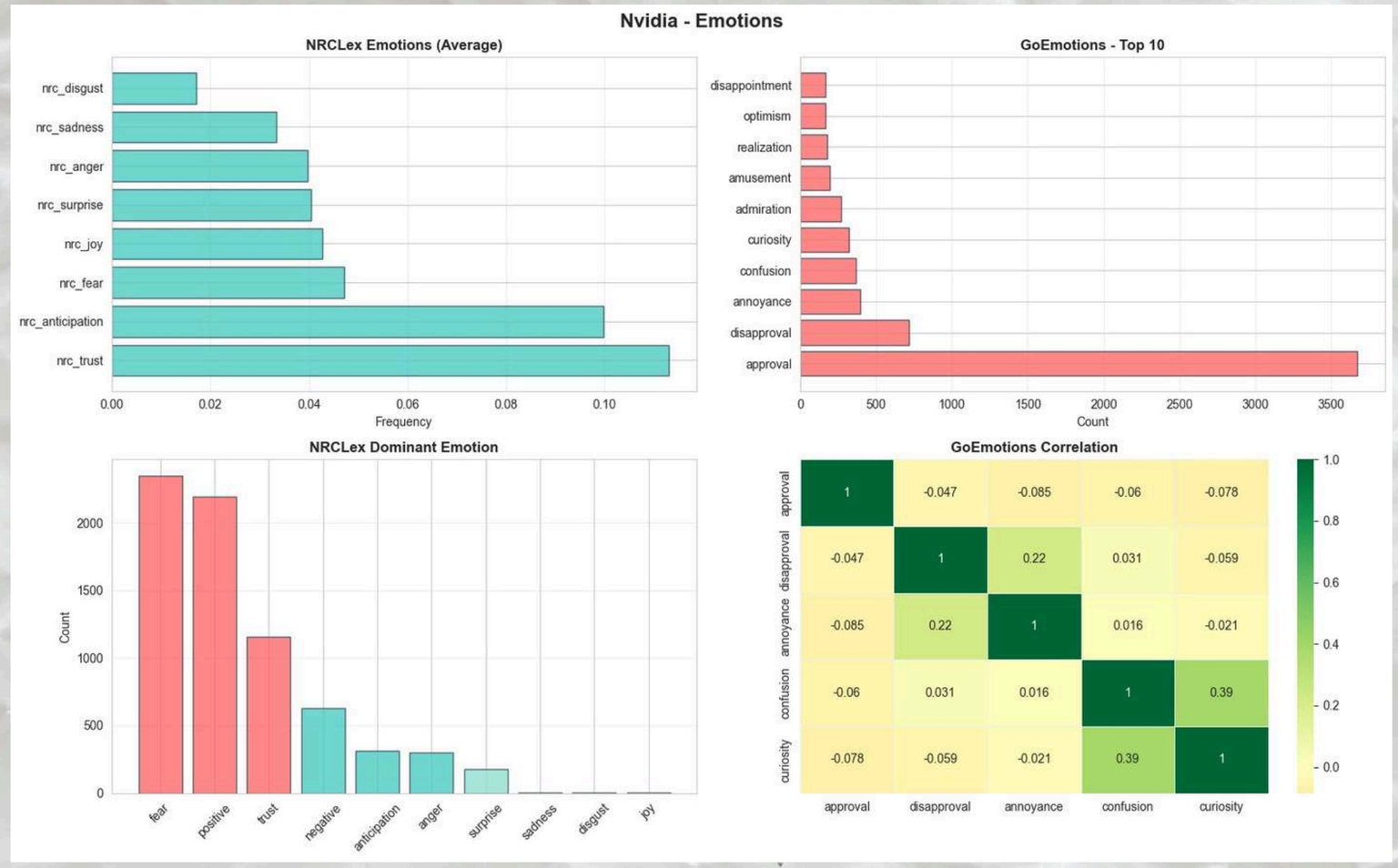
Sentiment analysis: Bitcoin emotions



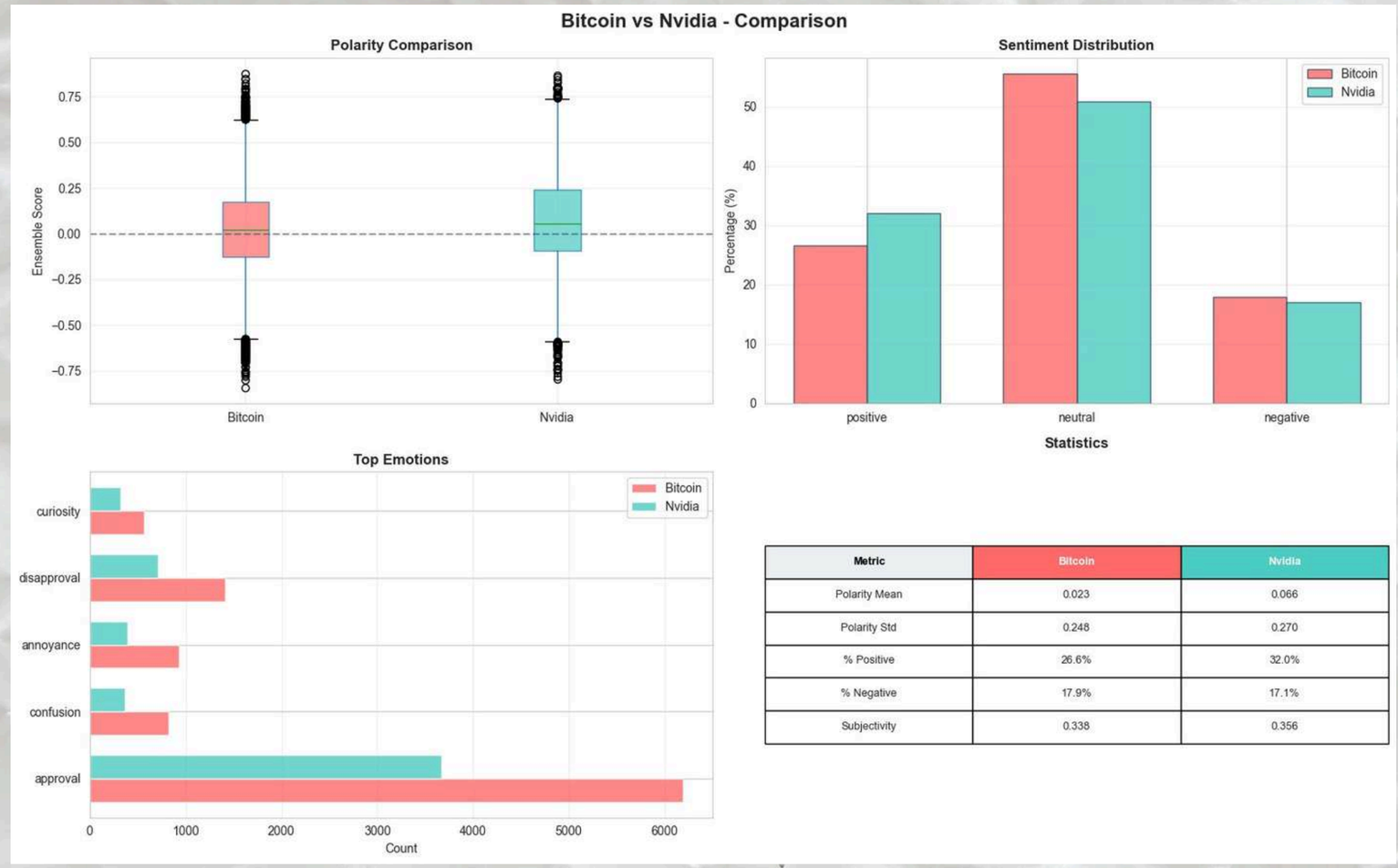
Sentiment analysis: Nvidia polarity



Sentiment analysis: Nvidia emotions

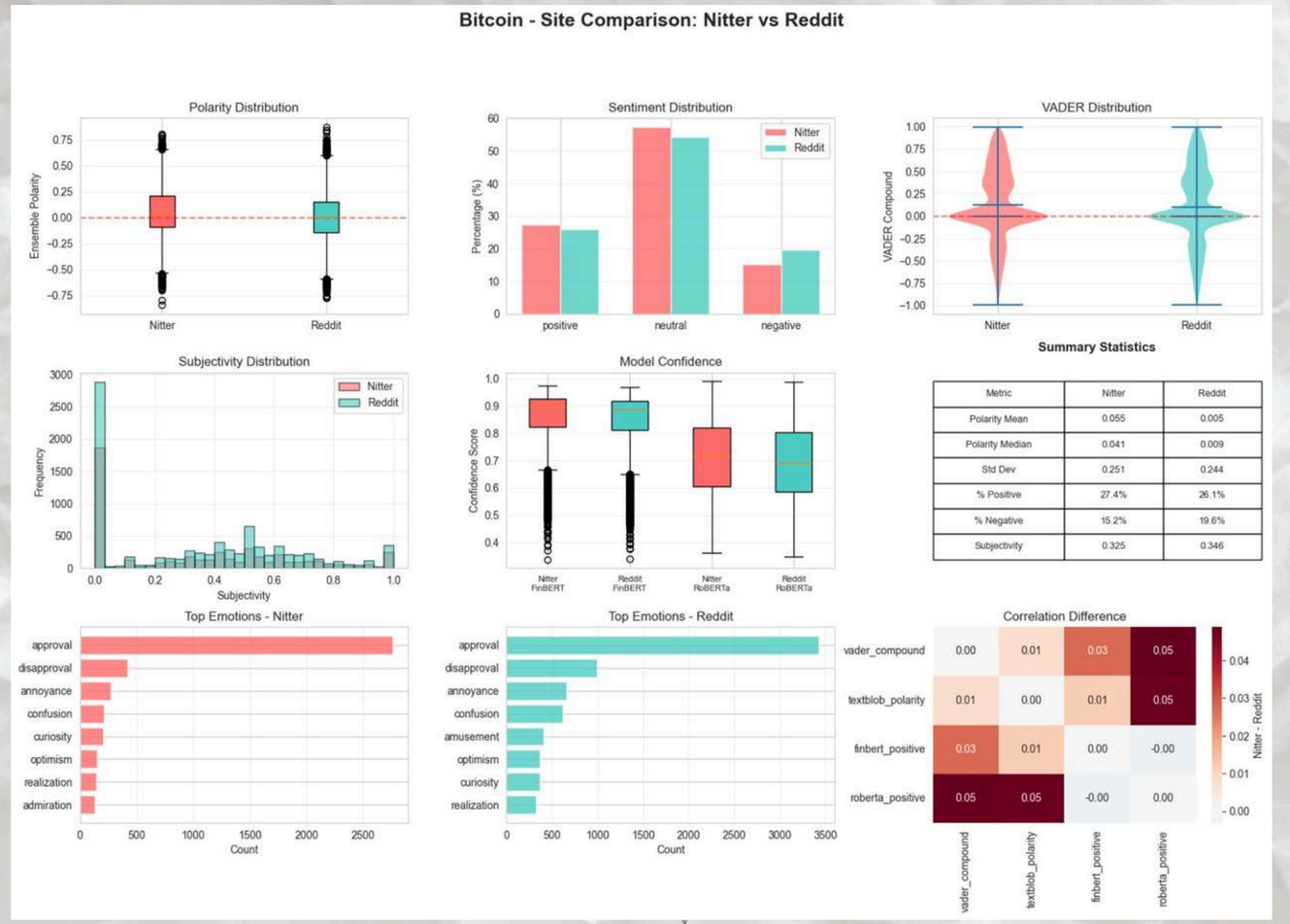


Sentiment analysis: Bitcoin & Nvidia comparison

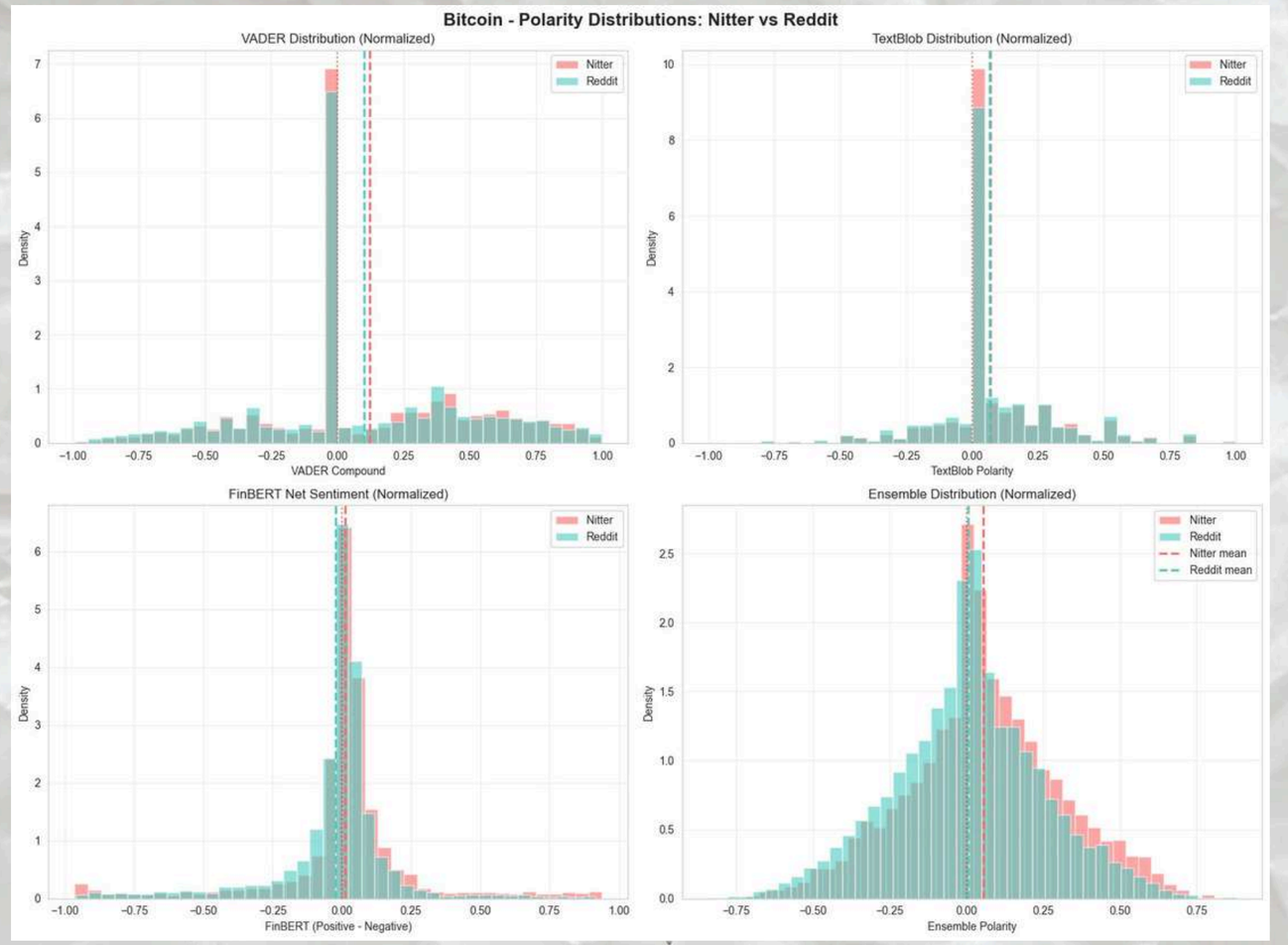


Metric	Bitcoin	Nvidia
Polarity Mean	0.023	0.066
Polarity Std	0.248	0.270
% Positive	26.6%	32.0%
% Negative	17.9%	17.1%
Subjectivity	0.338	0.356

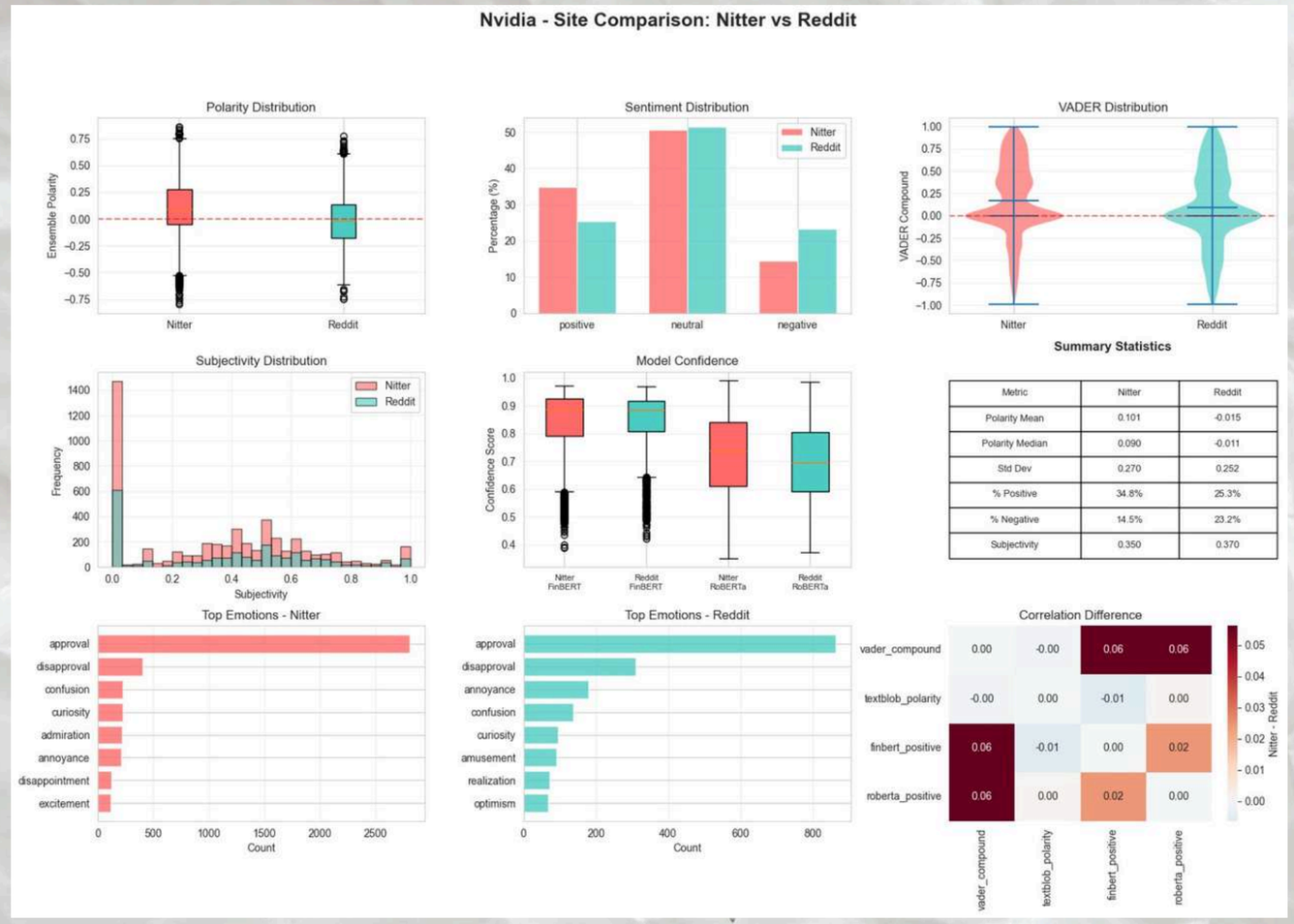
Sentiment analysis: Bitcoin site comparison



Sentiment analysis: Bitcoin site comparison

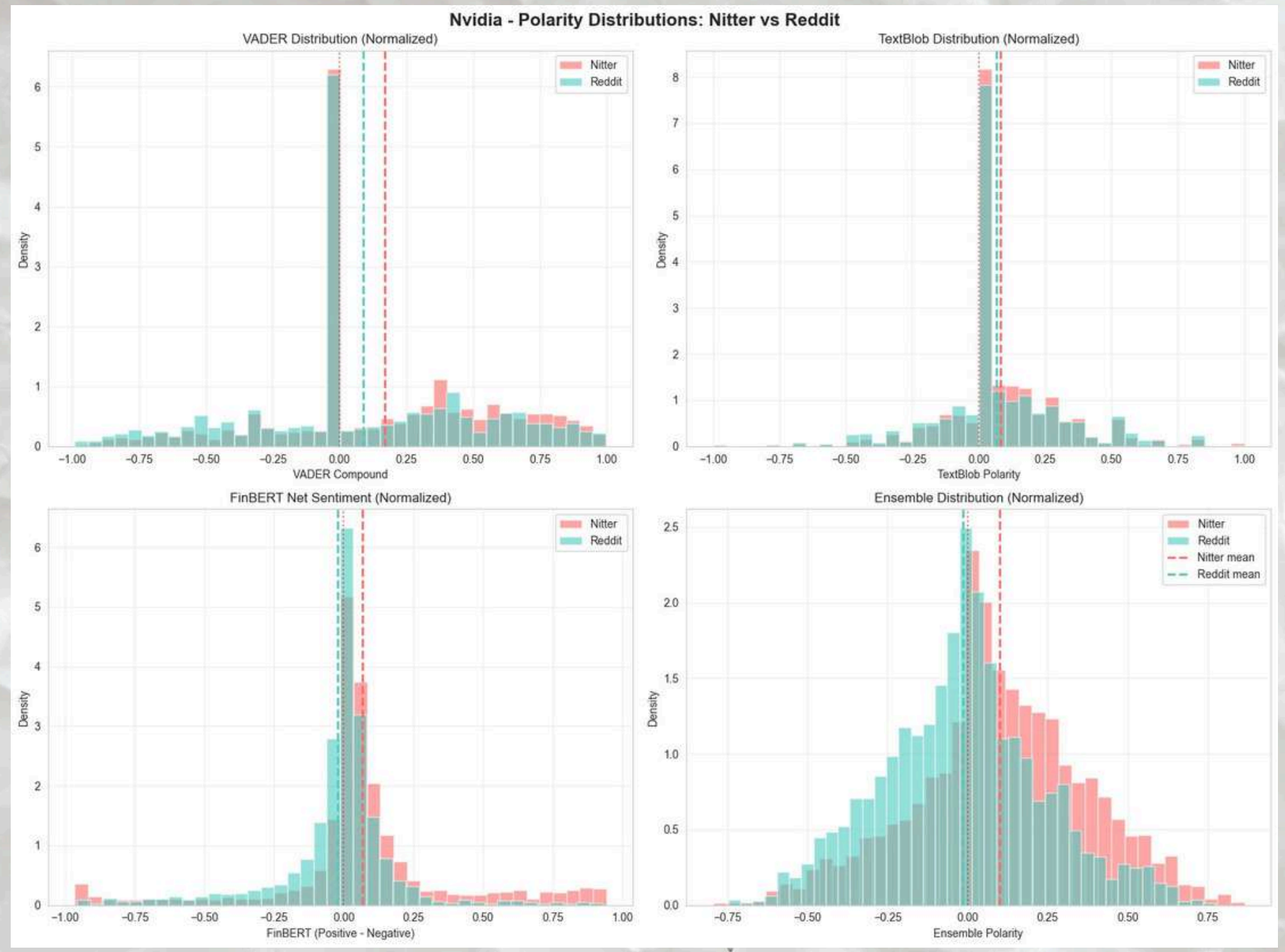


Sentiment analysis: Nvidia site comparison

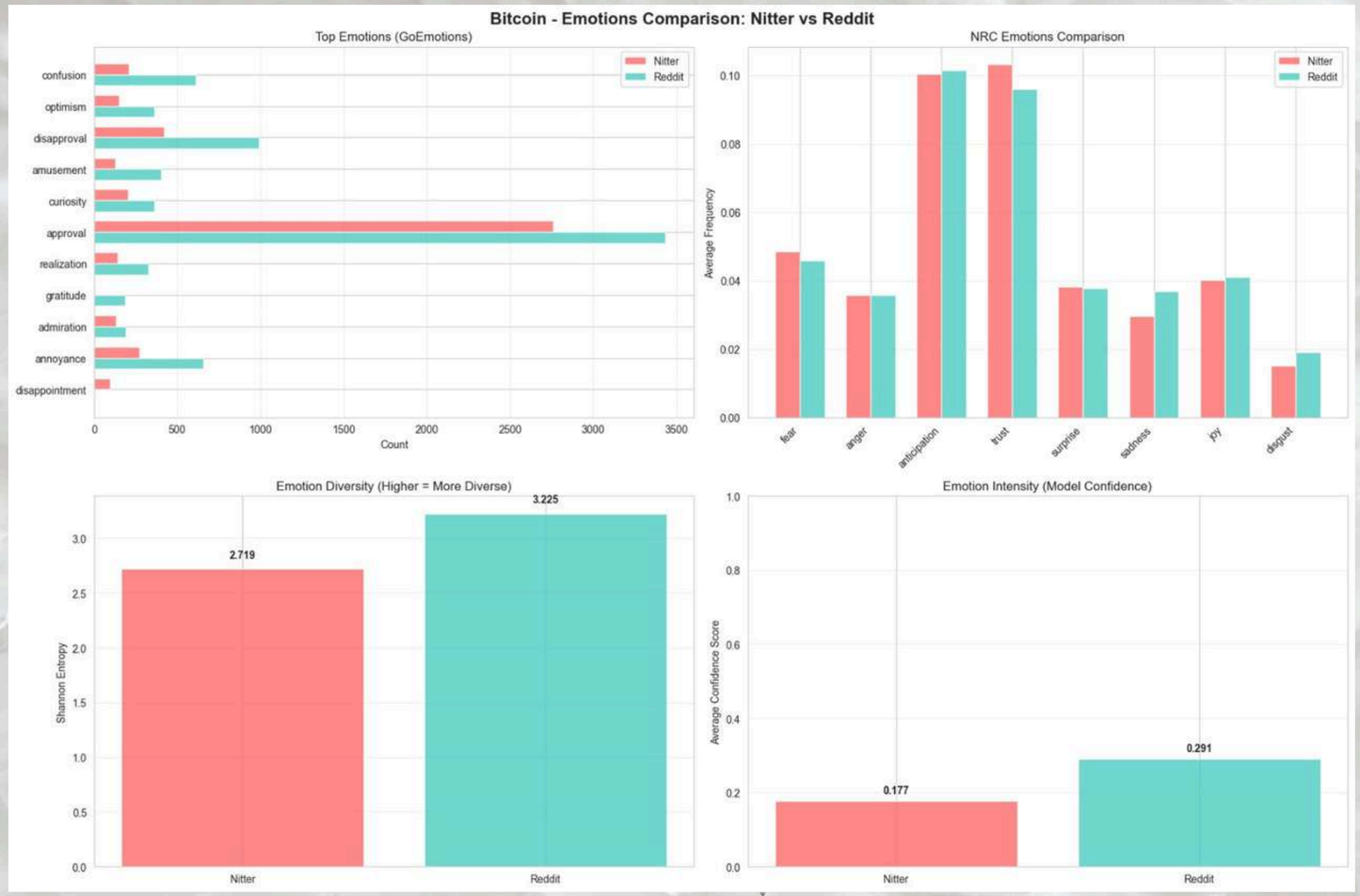


Metric	Nitter	Reddit
Polarity Mean	0.101	-0.015
Polarity Median	0.090	-0.011
Std Dev	0.270	0.252
% Positive	34.8%	25.3%
% Negative	14.5%	23.2%
Subjectivity	0.350	0.370

Sentiment analysis: Nvidia site comparison

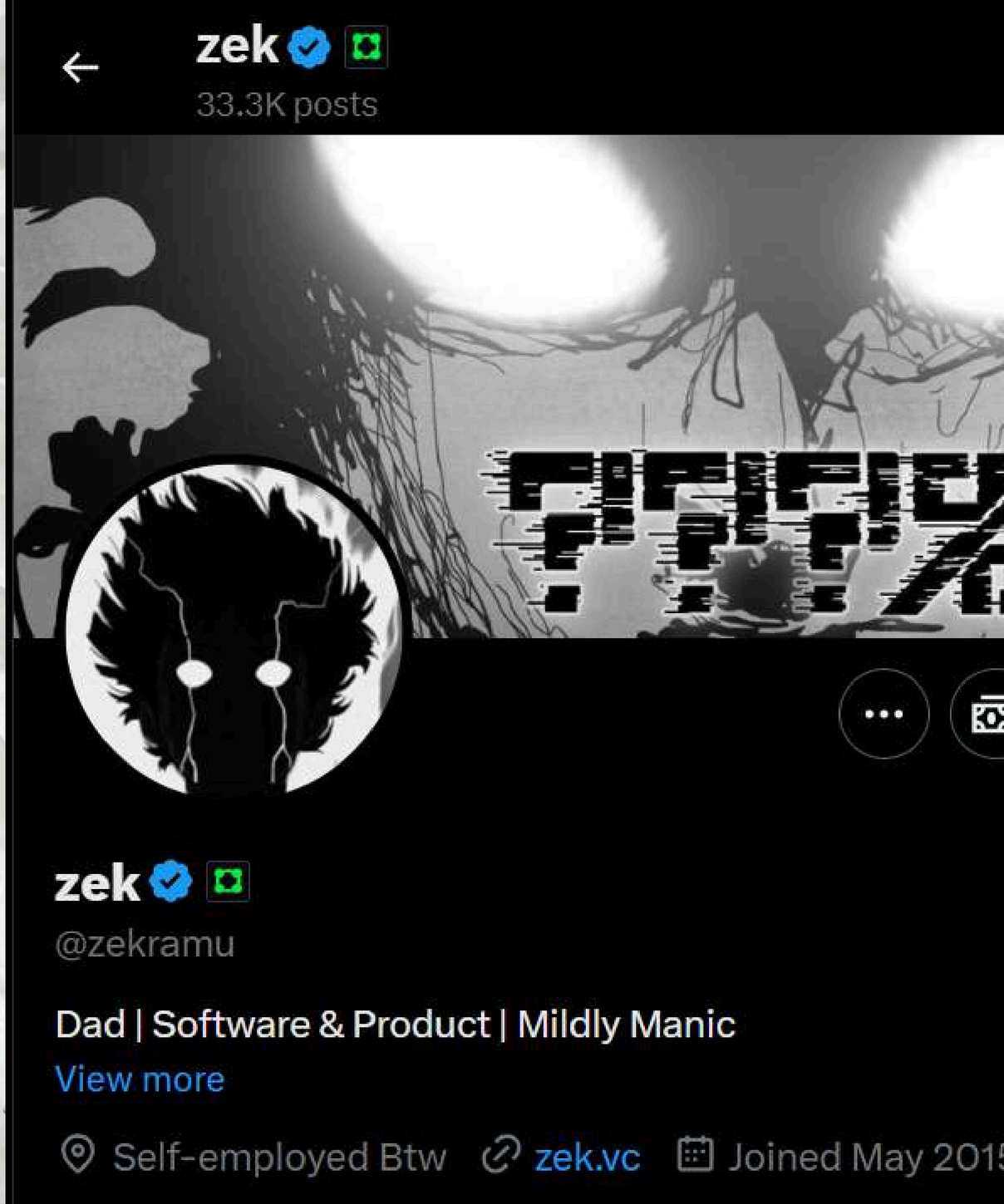


Sentiment analysis: Bitcoin emotion site comparison



Sentiment analysis: Nvidia emotion site comparison

Thank You


zek
33.3K posts
zek
@zekramu
Dad | Software & Product | Mildly Manic
View more
Self-employed Btw
zek.vc
Joined May 201